UNIVERSIDADE DE DO VALE DO TAQUARI - UNIVATES

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

CURSO DE ENGENHARIA DA COMPUTAÇÃO

APLICATIVO PARA NAVEGAÇÃO *INDOOR* UTILIZANDO A CÂMERA COMO SENSOR DE POSICIONAMENTO

Francisco Schwertner

Lajeado, novembro de 2017

Francisco Schwertner

APLICATIVO PRA NAVEGAÇÃO INDOOR UTILIZANDO A CÂMERA COMO SENSOR DE POSICIONAMENTO

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas da Universidade do Vale do Taquari - UNIVATES, como parte dos requisitos para a obtenção do título de bacharel em Engenharia da Computação.

Orientador: Fabrício Pretto

Lajeado, novembro de 2017

RESUMO

A popularidade dos dispositivos móveis, imensamente populares e equipados com sistemas de posicionamento global e conectividade com a Internet, propulsionaram o uso de serviços baseados em localização. Estes serviços promovem uma interação personalizada com o ambiente, com isso as pessoas podem encontrar a exata localização em um mapa, descobrir a melhor rota entre dois pontos, buscar e oferecer serviços e produtos próximos. Entretanto, os ambientes *indoor* ainda não possuem uma solução padrão para localização estabelecida, devido `as dificuldades de inferir o posicionamento no interior das construções. Alternativas utilizando técnicas de posicionamento visual, empregando a câmera do dispositivo, vêm sendo concebidas. Este trabalho analisou o uso da câmera de um *smartphone* como sensor de movimento em ambientes *indoor*, através da odometria visual. Além disso foram empregadas técnicas CBIR para reconhecimento da localização a partir de imagens. Um aplicativo para Android foi criado baseado no modelo de desenvolvimento híbrido. Os resultados dos testes relacionados a odometria visual mostraram que a técnica é capaz de captar o movimento do agente, porém o experimento não foi capaz de reproduzi-la com a precisão adequada. Contudo, foi possível verificar o desempenho do WebAssembly no processamento de imagens, desenvolvendo um algoritmo com a biblioteca OpenCV, portando o código C++ para *web*.

Palavras-chave: sistemas baseados em localização. Wayfinding. desenvolvimento híbrido. odometria visual. WebAssembly. CBIR

ABSTRACT

Mobile devices, largely popular and equipped with global positioning systems and Internet connectivity, have boosted the location-based services usage. These services promote a personalized interaction with the environment, thus people can find the exact location on a map, find the best route between two points, search for and offer services and products nearby. However, indoor environments don’t have an established standard solution yet, due to the difficulties of inferring the positioning inside the buildings. Alternatives using visual positioning techniques, employing the device's camera, have been designed. This work analyzed the use of a smartphone camera as a motion sensor in indoor environments through visual odometry. In addition, CBIR techniques were used to recognize the location from images. An Android app was created based on the hybrid development model. The tests results related to visual odometry showed that the technique was able to capture the movement of the agent, but it was not able to reproduce it with the proper precision. However, it was possible to verify the WebAssembly performance in image processing, developing an algorithm with the OpenCV library, porting the code C ++ to the web.

Keywords: location-based systems. Wayfinding. hybrid development. visual odometry. WebAssembly. CBIR

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

2D	Two-Dimensional
3D	Three-Dimensional
API	Application Programming Interface
BRIEF	Binary Robust Independent Elementary Features
CBIR	Content-Based Image Retrieval
CSS	Cascading Style Sheets
FPS	Frames Per Second
FAST	Features from Accelerated Segment Test
GPS	Global Postioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
LBS	Location-based service

MVP	Minimum Viable Product
ORB	Oriented FAST and Rotated BRIEF
RAM	Random Access Memory
RANSAC	Random Sample Consensus
SSD	Solid State Drive
REST	Representational State Transfer
RFID	Radio-Frequency IDentification
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
W3C	World Wide Web Consortium

SUMÁRIO

1 INTRODUÇÃO

Conforme Werner (2014) os desafios inerentes ao posicionamento e navegação sempre estiveram presentes na vida da sociedade. Atualmente, com a utilização de sistemas globais de navegação, com auxílio de satélites, a navegação em ambientes externos, se tornou algo comum. Dispositivos móveis equipados com GPS (Global Position System) impulsionaram o uso de serviços baseados em localização no mundo inteiro.

Segundo Springer (2012), a navegação e localização *indoor*, no interior de construções, com auxílio de dispositivos móveis, é um tema com bastante evidência no mundo da computação. Uma vez que as pessoas passam a maior parte do seu tempo em ambientes fechados, faz grande sentido a utilização de mecanismos que facilitem, agilizem a locomoção e enriqueçam a experiência do usuário dentro destes ambientes (SUDHAKARAN, 2014).

Lugares públicos como shoppings, hospitais e universidades podem fornecer uma maneira fácil para que os seus frequentadores encontrem direções para lojas, banheiros, salas de aula, localização de pessoas, monitoramento e rastreamento da movimentação. Uma das grandes dificuldades encontradas em áreas fechadas é a falta de um sistema de posicionamento global, diferentemente dos ambientes abertos, que podem fazer uso de sistemas como o GPS (WERNER, 2014).

De acordo com Springer (2012), técnicas vêm sendo desenvolvidas para ambientes internos com objetivo de fornecer o posicionamento similar ao GPS. É possível destacar atualmente as tecnologias *wireless*, que utilizam Wi-Fi *fingerprinting*, *Bluetooth* ou RFID (Radio-Frequency IDentification), para estimar a posição baseando-se na triangulação e proximidade do sinal (MAUTZ, 2012). Hu (2015) mostra que um grande número de pesquisas baseiam-se na utilização de marcos visuais, que são extraídos das imagens, e possibilitam o seu reconhecimento. A implementação consiste em capturar imagens do ambiente e comparar com imagens previamente registradas, que contenham as coordenadas de referência (HU, 2015).

Szelinski (2011) cita que a computação visual é a área que busca descrever o mundo com base em imagens e reconstruir as suas propriedades de forma, iluminação e cor. Algumas de suas aplicações são o reconhecimento de lugares baseado em imagem e a detecção de movimentos. A utilização da câmera como sensor de movimentação é o principal tema da odometria visual. Comumente usada na área da robótica, a odometria visual tenta traçar a posição e trajetória de um agente, analisando as imagens obtidas de câmeras (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Técnicas de busca, em que a imagem possa ser pesquisada diretamente, utilizando as informações presentes nela mesmo ao invés de suas descrições textuais, vêm ganhando espaço. Neste campo vem se destacando o conceito de CBIR (Content-Based Image Retrieval), envolvendo as tecnologias que auxiliam a organização das imagens digitais, baseadas no seu conteúdo visual (BHARTI; WADHWA, 2013).

O desenvolvimento de aplicativos para dispositivos móveis, empregando um modelo híbrido, surgiu para facilitar a construção e manutenção de aplicativos *mobile* multiplataforma (LOPES, 2016). Com o atual poder de processamento dos dispositivos móveis, a possibilidade de executar algoritmos de visão computacional, no lado do cliente, tornou-se viável. Muitas aplicações complexas podem agora ser realizadas no navegador de Internet, sem a necessidade de instalações extras (AKHMADEEV**,** 2015). Recentemente os representantes dos quatro navegadores

mais populares (Chrome, Microsoft Edge, Firefox e WebKit/Safari), entraram em um consenso sobre o WebAssembly. Esta nova tecnologia, que já está presente nos navegadores, deve permitir a execução de aplicações na web com o desempenho similar às aplicações nativas (CLARK, 2017a).

O trabalho apresentou uma proposta prática para o problema da navegação *indoor,* na tentativa de oferecer aos frequentadores dos espaços internos, uma forma inteligente de *wayfinding.* Um aplicativo para *smarphones* Android foi desenvolvido através do modelo híbrido, utilizando somente a câmera como sensor de movimento. Com isto foi possível fazer uma análise sobre a efetividade do algoritmo de odometria visual e demonstrar uma aplicação prática das técnicas CBIR. Foi possível também verificar o desempenho do WebAssembly, para processamento de imagens, explorando a eficiência desta nova tecnologia, portando o código escrito em C++ para web.

1.1 Objetivo geral

O presente trabalho teve como objetivo geral desenvolver uma solução computacional que auxilie no processo de localização em ambientes internos, com base na aquisição de imagem da câmera de um *smartphone*. Com isto foi possível fazer uma análise sobre a efetividade do algoritmo de odometria visual, empregando a homografia, bem como demonstrar uma aplicação prática das técnicas CBIR. Além disso, foi possível explorar o desenvolvimento e o desempenho do WebAssembly, para processamento de imagens, portando o código escrito em C++ para web.

1.2 Objetivos Específicos

Como objetivos específicos do trabalho podem ser citados:

a) Pesquisar bibliografias que abordem os assuntos relacionados ao desenvolvimento da aplicação proposta;

b) Identificar, desenvolver e aplicar as teorias relacionadas a odometria visual monocular;

c) Analisar o desempenho do WebAssembly e portar o algoritmo de processamento de imagens, escrito em C++, empregando a biblioteca OpenCV, para web;

d) Explorar as funcionalidades dos sistemas CBIR;

e) Criar a arquitetura do aplicativo, projetar a usabilidade e as funcionalidades necessárias, seguindo o modelo de desenvolvimento *mobile* híbrido;

1.3 Motivação

Atualmente os espaços *indoor* não possuem uma solução de posicionamento estabelecida, nestes ambientes não é possível contar com sinal GPS. Este trabalho tem como motivação principal propor um sistema de localização *indoor*, empregando uma tecnologia de posicionamento que não precise de investimento para implantação. Outro ponto de interesse é a comprovação de que a odometria visual pode ser implementada em um dispositivo móvel, utilizando WebAssembly, portando as funcionalidades da biblioteca OpenCV (OPENCV, 2017a) para *web*.

1.4 Estrutura do Trabalho

O primeiro capítulo faz uma introdução sobre o trabalho em linhas gerais, demonstrando a importância dos assuntos envolvidos no campo da localização *indoor*. São apresentados os objetivos do trabalho, justificativa e a estrutura do estudo. O segundo capítulo compõe-se de um estudo bibliográfico, onde os assuntos teóricos são revisados, a fim de situar e esclarecer o leitor quanto as matérias que nortearão o trabalho. Pretende-se explicar de forma clara os aspectos relacionados aos sistemas de localização, posicionamento, *wayfinding*, visão computacional e odometria visual. Além disso, o segundo capítulo apresenta alguns conceitos relacionados as tecnologias empregadas no trabalho, como os sistemas

de localização baseada em imagem, construção de aplicativos híbridos e o novo formato WebAssembly.

O capítulo três busca elencar os trabalhos relacionados a localização visual *indoor*, utilizando dispositivos móveis. No capítulo quatro são abordas a metodologia e o funcionamento do experimento, delimitando o cenário de testes e as atividades realizadas. O quinto capítulo detalha e discute os resultados obtidos. Finalizando, no capítulo seis, são apresentadas as conclusões e considerações finais, juntamente às propostas para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesse capítulo, pretende-se realizar uma revisão da literatura, dos assuntos que oferecem suporte ao desenvolvimento do experimento proposto. Primeiramente serão tratados os aspectos relevantes aos sistemas de localização, com apresentação dos sistemas de posicionamento utilizados atualmente em ambientes *indoor.* Posteriormente *s*erão introduzidos os conceitos e aplicações relacionados a computação visual e odometria visual, com o objetivo de demonstrar as técnicas aplicadas a estimativa de trajetória, empregando a câmera como sensor de movimento. Na parte final serão apresentados o modelo de desenvolvimento *mobile* híbrido e as bibliotecas JavaScript utilizadas para computação visual no ambiente *web,* além de embasar o leitor a respeito do novo formato WebAssembly.

2.1 Sistemas baseados em localização

Um LBS (Location-Based System), é um sistema que provê funcionalidades específicas, baseado na localização da entidade portadora do dispositivo móvel (WERNER, 2014). Exemplos deste tipo de aplicação incluem serviços de emergência, sistemas de navegação, sistemas colaborativos, guia de turismo e rede social (SCHILLER; VOISARD, 2004).

Um dos fatores que impulsionaram o desenvolvimento de aplicações LBS foi o aumento na venda de *smartphones*, onde em 2010 o número de dispositivos móveis vendidos já ultrapassava o de *desktops* (EXAME, 2010). Em maio de 2016 o número de *smartphones* em uso no Brasil chegava a casa dos 168 milhões, conforme a 27º Pesquisa Anual de Administração e Uso da Tecnologia da Informação nas Empresas, realizada pela fundação Getúlio Vargas de São Paulo (FOLHA, 2016).

A junção destes fatos cria a oportunidade para desenvolvimento e adoção dos sistemas baseados em localização, que fazem uso de dispositivos móveis, redes de comunicação *wireless* e sistemas de posicionamento, para entregar e gerar valor para os usuários. A possibilidade de criar uma interação personalizada vem atraindo a atenção do mercado. Atualmente, companhias como Google e Facebook utilizam dados de localização para recomendação de produtos, serviços e eventos, com o objetivo de melhorar a experiência do usuário e propiciar maior interação nas redes sociais (FACEBOOK, 2015).

Com auxílio dos sistemas baseados em localização é possível encontrar a exata localização em um mapa, descobrir a melhor rota entre dois pontos, buscar e oferecer serviços e produtos próximos. Comumente estes tipos de sistemas são empregados em ambientes *outdoor*, e utilizam para isso serviços como OpenStreetMaps (OPENSTREETMAP, 2017) e Google Maps (GOOGLE, 2017), entre outros, para disponibilizar o mapeamento. O posicionamento é geralmente obtido com uso de GPS ou triangulação Wi-Fi (SPRINGER, 2012).

Os sistemas de localização vêm sendo utilizados desde a década de 70, quando o departamento de defesa dos Estados Unidos passou a utilizar sua infraestrutura de satélites global para encontrar a posição de pessoas e objetos. Em 1980 o governo americano decidiu liberar para fins não militares. Somente na década de 90, com o advento das redes móveis, houve um grande salto no uso dos serviços de localização (SCHILLER; VOISARD, 2004).

Analogamente, há a existência dos ambientes *indoor*, no interior das construções. Shoppings, hospitais, museus e prédios públicos necessitam de uma forma para informar as direções aos usuários, mostrar o caminho que leva a

determinada sala, encontrar banheiros e escadas, lojas ou obras de arte. Projetos para resolver estas questões vêm sendo desenvolvidos, testados e implementados. A maior diferença entre os sistemas *indoor* e *outdoor* são as dificuldades técnicas apresentadas no âmbito *indoor*. Não existe no momento uma solução padrão para disponibilização dos serviços de mapeamento, posicionamento e localização, nos mesmos moldes com que acontece no meio *outdoor* (SPRINGER, 2012).

Grandes empresas têm mostrado interesse crescente no mercado de navegação *indoor.* Em 2014 a Apple adquiriu a empresa WIFISlam, especialista em navegação *indoor*, por aproximadamente U$$ 20 milhões. Outro investimento foi a empresa PrimeSense 3D, por um valor informado de U$$ 360 milhões. Hoje as novas aplicações baseadas em localização estão limitadas apenas pela criatividade dos desenvolvedores, empresas vêm investindo constantemente no setor, a empresa de pesquisa ABI Reserach estima que o mercado de mapeamento *indoor* valerá cerca de US$ 4 bilhões até 2018 (SUDHAKARAN, 2014).

De acordo com Werner (2014, p.5):

> os serviços de localização envolvem três aspectos básicos: a habilidade de inferir a localização, a habilidade de comunicar a informação e a habilidade de utilizar a informação com objetivo de prover o serviço.

Com o objetivo de delimitar e entender as tarefas dentro de um LBS, é comum a divisão entre as seguintes características (WERNER, 2014):

a) Terminal: dispositivo móvel capaz de se movimentar no ambiente

b) Localizador: sistema que habilita a inferência da localização do dispositivo móvel;

c) Provedor de serviço: serviço que manipula e explora as informações de localização coletadas e gera valor adicional;

d) Usuário do serviço: usuário utilizador do serviço;

É possível também fazer uma classificação quanto ao modo de operação. No modelo orientado a pessoa, as aplicações são voltadas ao usuário, onde a

localização é utilizada de acordo com as suas necessidades, um exemplo é a busca por amigos próximos. Já no modelo orientado ao dispositivo, normalmente os usuários não controlam o serviço nem o dispositivo, pode-se citar como exemplo o rastreamento de carros (SCHILLER; VOISARD, 2004).

BUCZKOWSKI (2012), ASHWINI e USHA (2014) compartilham uma ideia semelhante quanto aos componentes da infraestrutura, necessários a disponibilização um serviço LBS. A Figura 1 expõem visualmente os componentes do sistema.

Figura 1 - Componentes de uma arquitetura LBS

Fonte: Adaptado de (BUCZKOWSKI, 2012)

1) Sistema de posicionamento: sistema que permite a localização tanto *outdoor* como *indoor*. Satélite, rastreio celular, RFID, *Bluetooth* e redes Wi-Fi;

2) Rede de Comunicação: utilizada para transmitir dados entre dispositivos e o servidor;

3) Serviço de aplicação: responsável por analisar os dados de localização e retornar informações úteis aos usuários;

4) Provedor de conteúdo: órgãos do governo, empresas de comércio e indústria que disponibilizam os dados e mapas;

5) Dispositivos móveis: qualquer dispositivo que possa utilizar os componentes mencionados acima;

6) Usuário: portador do dispositivo, que recebe as informações úteis em relação a sua localização.

Quanto as técnicas utilizadas para o posicionamento, em relação ao local de operação, os serviços de localização são divididos em *outdoor* e *indoor*. O sucesso das aplicações criadas para o meio *outdoor* tem gerado grande interesse, empresas e pesquisadores tentam estender estas funcionalidades para o meio *indoor*. Entretanto, vários desafios precisam ser ultrapassados, a complexidade para gerar e manter os mapas internos dos prédios de uma forma automatizada e a maneira para obter a posição no interior das construções, são alguns dos desafios (WERNER, 2014).

2.2 Posicionamento *Indoor*

Segundo Werner (2014, pg. 73) ”o posicionamento *indoor* é a tarefa de inferir a localização de um dispositivo móvel dentro de um prédio”. No processo de determinar a localização, dois termos são utilizados constantemente e de forma similar, posicionamento e localização. Posicionamento é o comumente utilizado para descrever a posição da entidade, com precisão, e indicar a movimentação em direção a certo local. Já a localização é associado estimativa do local onde a entidade se encontra, onde a precisão exata não é tão importante ou até indisponível (MAUTZ, 2012).

Os mecanismos para posicionamento *indoor* sofrem com a falta de precisão. Os cálculos a fim de estimar a exata posição do objeto sofrem com a calibração inadequada dos sensores de medição. Além disso, a infraestrutura construída, para auxiliar no serviço de posicionamento sofre com interferências. É importante mencionar também a presença de diferentes elementos dentro da construção,

inclusive a presença de outras pessoas, que podem interferir nas medições captadas pelos sensores (STOJKSKA; KOSOVIC; JAGUST, 2016).

Ainda assim, apesar dos obstáculos, as tecnologias voltadas à localização *indoor* estão evoluindo. Com o aperfeiçoamento dos equipamentos e técnicas, a precisão vêm aumentando, proporcionando novas aplicações. É possível destacar atualmente as tecnologias *wireless*, que utilizam Wi-Fi *fingerprinting*, *Bluetooth* ou RFID, para estimar a posição baseando-se na triangulação e proximidade do sinal (MAUTZ, 2012).

Há outras soluções desenvolvidas recentemente, utilizadas em menor escala, como a localização magnética, que utilizam o sensor bússola, presente em dispositivos móveis atuais (KARIMI, 2015). A empresa IndoorAtlas é uma das pioneiras neste tipo de implementação, que é capaz de reconhecer as variações no campo magnético da terra, causadas pelas distorções e interferências, geradas pelas estruturas metálicas, presentes nas construções. Estas variações criam um tipo de impressão digital do prédio, o que possibilita calcular a movimentação do dispositivo (STERLING, 2014).

É possível encontrar também soluções que se baseiam na utilização de sensores IMU (*Inertial Measurement Unity*). A grande maioria dos dispositivos atuais possuem uma unidade de medição interna IMU, que consiste em um conjunto de acelerômetros e giroscópios e em alguns casos magnetrômetros, capazes de medir o ângulo, direção e força do movimento. A técnica chamada de Sistema de Navegação Inercial utiliza estes sensores para computar a posição, velocidade e orientação do dispositivo. O processo para calcular a posição atual é sempre baseado na posição anterior, o que pode gerar desvios, de acordo com a quantidade de interferência aferida pelos sensores (MAUTZ, 2012).

Iniciativas recentes têm demonstrado a combinação de várias tecnologias para determinar o posicionamento *indoor*. A fusão dos dados coletados pelos sensores é utilizada de uma forma complementar. Liu et al (2015) propõe uma solução híbrida para *smartphones*, utilizando técnicas de posicionamento Wi-Fi e sensores IMU. Kanaris et al (2017) apresentam um sistema baseado em marcadores *bluetooth* e mapas de sinal Wi-Fi. Wang et al (2016) mostram um método que

mescla o reconhecimento dos passos da pessoa que está portando o *smartphone*, empregando os sensores IMU e a detecção da proximidade dos *access points* Wi-Fi.

A popularidade e o aumento na capacidade de processamento dos *smartphones*, em combinação com as câmeras integradas cada vez mais precisas, vêm tornando viáveis o aparecimento das técnicas visuais de posicionamento. Um grande número de pesquisas baseiam-se na utilização de marcos visuais, que são extraídos das imagens e possibilitam o seu reconhecimento. A implementação consiste em capturar imagens do ambiente e comparar com imagens previamente registradas, que contenham as coordenadas de referência (HU, 2015).

2.3 *Wayfinding*

O termo *wayfinding* foi introduzido no final da década de 70, com intuito de conceitualizar os estudos relacionados a compreensão da movimentação das pessoas e a interação com o ambiente. Para Athur e Passini (1992) o *wayfinding* procura entender a movimentação das pessoas e a sua relação com o espaço, a tendência do homem em entender onde está, para onde ir, e como obter o melhor caminho entre o início e o fim do destino. Adicionalmente, O'Grady J. V. e O'Grady K. V. (2008, p. 72) complementam o significado de *wayfinding* "como um indivíduo se orienta em um ambiente desconhecido, e os processos cognitivos usados para determinar e seguir uma rota ou travessia de um ponto ao outro".

Conforme Satalich (1995) o processo de *wayfinding* ocorre em quatro etapas:

- **Orientação:** o usuário deve ser capaz de reconhecer sua localização em relação aos objetos vizinhos e o destino;
- **Escolha da rota:** relaciona-se a escolha do caminho que conduz ao destino desejado, deve-se preferir a indicação de caminhos curtos possíveis;
- **Observação da rota:** observação em tempo real da trajetória, para confirmação de que o caminho está sendo seguido corretamente;

- **Reconhecimento do destino:** para confirmação da chegada ao destinho final é imprescindível identificá-lo.

Karimi (2015) descreve as diferenças entre navegação e *wayfinding*, frenquentemente utilizados como mesmo sentido. O *wayfinding* refere-se ao estabelecimento de rotas entre dois locais diferentes. Já a navegação consiste em receber notificações em tempo real, com orientações que servem como guia, para percorrer a rota determinada.

Conforme Foltz (1998), são utilizados três critérios para definir a navegabilidade de um lugar. Primeiro deve ser possível inferir a localização inicial, segundo, definir se a rota para o destino final pode ser encontrada, por fim deve-se verificar a qualidade das informações presentes no ambiente e que auxiliam no processo de *wayfinding*. Os princípios envolvidos no *wayfinding*, a fim de melhorar a navegação a partir das informações do ambiente, são descritos por Foltz (1998):

a) Criar uma identidade em cada local: é preciso dar a cada espaço navegável uma identidade única capaz de distingui-lo dentro do ambiente.

b) Utilizar pistas de orientação: locais precisam ser de fácil reconhecimento e memorização.

c) Criar caminhos bem estruturados: os caminhos devem ser contíguos e apresentar o progresso ao percorrê-lo.

d) Criar regiões visualmente diferenciáveis: dividir as regiões do ambiente com uma identidade visual distinta entre si.

e) Restringir ao usuário as opções de navegação: estipular de preferência somente um caminho principal entre início e destino.

f) Utilizar pesquisa visual: mostrar um mapa que mostre o ponto de vista do indivíduo, com isso a tarefa de saber o que há ao seu redor e destinos próximos e a distância até o destino.

g) Fornecer sinais que auxiliem a decisão: criar pontos onde o indivíduo precisará tomar uma decisão, seguir em frente ou mudar de direção.

2.4 Visão Computacional

A visão humana é um sistema refinado, capaz de sentir e agir de acordo com estímulos visuais, seu estágio atual é resultado de milhões de anos de evolução (NIXON; AGUADO, 2012). Da mesma forma que o homem vê e visualmente sente o ambiente, a visão computacional é a ciência que tem como objetivo principal, dotar computadores e outras máquinas com visão ou a habilidade de ver (RYAN, 2012). A computação visual foca na extração, análise e compreensão de informações úteis, a partir de imagens (BMVA, 2017).

Apesar de a visão computacional tentar recriar as funcionalidades da visão humana, ainda não é possível reconstituir exatamente todas as peculiaridades existentes na atividade da percepção visual humana (NIXON; AGUADO, 2012). A complexidade é relacionada à percepção do mundo real, o processo de entender as informações visuais, buscar significados e estabelecer modelos físicos e probabilísticos que descrevam a cena. Descrever o mundo com base em imagens, e reconstruir as suas propriedades de forma, iluminação e cor pode parecer uma tarefa simples para humanos e animais, mas para os computadores ainda é uma atividade sujeita a falhas (SZELISKI, 2011).

Várias aplicações têm sido desenvolvidas nos últimos anos, de acordo com Szielinski (2011):

- Reconhecimento de caracteres
- Inspeção de produtos manufaturados na indústria e controle de qualidade
- Construção de modelos 3D a partir de imagens coletadas da superfície dos objetos
- Análise de exames médicos
- Veículos autônomos
- Detecção e captura de movimento

- Reconhecimento biométrico

A computação visual pode ser comparada inversamente com a computação gráfica. Na computação gráfica são produzidas imagens a partir de modelos 3D, enquanto que a computação visual procura encontrar modelos com base em imagens. Por ser um campo de estudo relativamente novo, nos anos 70 surgiram os primeiros estudos relativos ao assunto. Não existe no momento uma abordagem padrão de como os problemas devem ser resolvidos (RYAN, 2012).

Por ser uma área de conteúdo extenso e interdisciplinar, é comum a divisão em subdomínios de pesquisa, dentre os quais é possível citar a reconstrução de cena, detecção de eventos, rastreamento de vídeo, reconhecimento de objetos, estimativa de movimento, indexação, aprendizado e restauração de imagens. Em grande parte das aplicações, as funções envolvidas nos sistemas de computação visual, podem ser divididas em: aquisição da imagem, pré-processamento, extração de características, detecção/segmentação, processamento alto nível e reconhecimento de imagem (RYAN, 2012).

A visão baseia-se primordialmente na aquisição de imagens. A qualidade da obtenção depende diretamente da iluminação, para gerar imagens detalhadas e com contraste, indispensável a eficácia dos algoritmos de visão computacional (DAVIES, 2012). O pré-processamento é empregado para equalizar, corrigir e aperfeiçoar aspectos da imagem. Nesta etapa acontece correção na iluminação, desfocagem, redimensionamento da imagem, entre outros (KRIG, 2014).

No estágio de extração de caraterísticas são detectados os atributos que se destacam na imagem, pontos dos cantos de objetos ou linhas, que podem ser utilizados para estimar movimento e reconhecer imagem (KRIG, 2014). Em alguma parte do processo, para delimitar a área de interesse, é comum segmentar parte da imagem. Nas duas últimas etapas, processamento alto nível e reconhecimento, os algoritmos normalmente trabalham apenas com pontos específicos das imagens, para determinar modelos probabilísticos, estimar movimento ou o reconhecimento dos objetos (RYAN, 2012).

2.5 Odometria Visual

Odometria visual compreende o processo de calcular a posição e recriar a trajetória de um agente (humano, veículo, robô), a partir de um fluxo de imagens captados utilizando uma ou mais câmeras. O funcionamento baseia-se na estimativa incremental da posição, levando em conta as mudanças que a movimentação provoca na imagem. Para o seu funcionamento é essencial uma iluminação adequada e quantidade suficiente de texturas do ambiente. Além disso, o fluxo de imagens captadas ou *frames* precisa ter um mínimo de sobreposição, para permitir a comparação da cena (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

A odometria é muito utilizada no campo da robótica, onde um dos principais desafios é inferir com precisão a localização do veículo em relação ao tempo. Uma das alternativas mais simples é a odometria baseada no giro da roda. No entanto, esta técnica sofre com o erro acumulado, em virtude de desníveis no terreno ou deslize das rodas (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011). A odometria visual não é suscetível ao terreno, por este motivo foi utilizada pela NASA, em duas missões de exploração a Marte (NASA, 2005). Além disso, a odometria visual é mais barata e precisa do que técnicas baseadas em GPS, sensores IMU, odometria baseada no giro da roda, com um erro relativo variando de 0.1% a 2% (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Para resolução dos problemas inerentes a odometria visual, é comum a distinção entre três abordagens para extração do movimento, manipulando o fluxo de imagens: o método baseado na extração de características, que utiliza um conjunto de pontos correlatos entre imagens consecutivas; o método baseado na aparência, que determina a movimentação de câmera a partir do gradiente de intensidade dos *pixels* da imagem; e o método híbrido, que utiliza uma combinação de extração de características e aparência (AQEL et al., 2016) .

2.5.4 Odometria visual monocular e estéreo

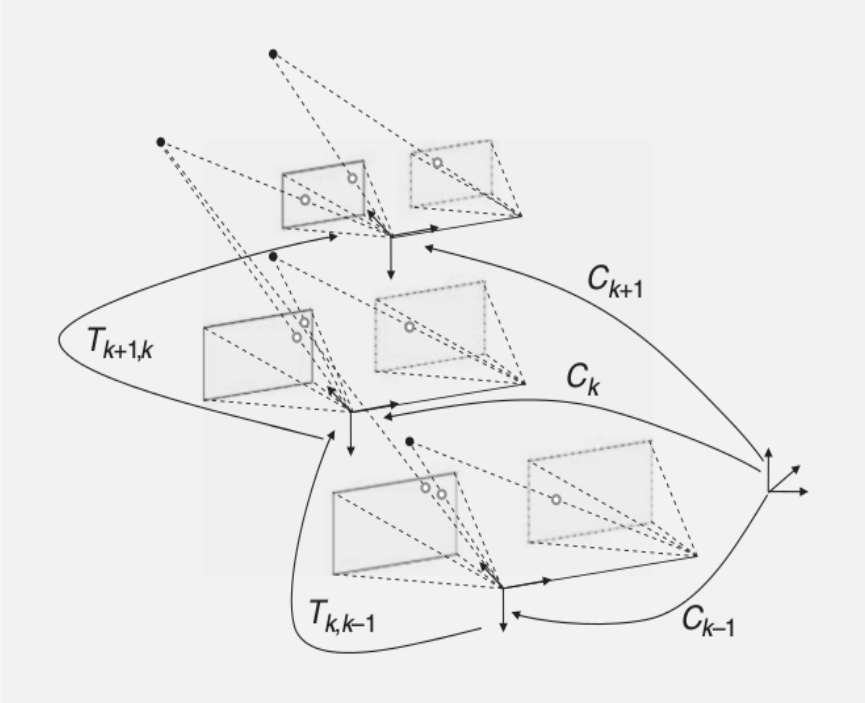
A maioria dos métodos propostos utiliza uma ou duas câmeras para aquisição das imagens, sendo classificados como odometria visual monocular e estéreo respectivamente. Na odometria visual estéreo, a estimativa de profundidade e escala são possíveis analisando apenas um único quadro da imagem, através de um processo de triangulação onde a distância entre as câmeras é conhecido. Com isso é possível calcular o deslocamento em unidades de medida, como centímetros ou metros (AQEL et al., 2016).

Entretanto, o uso de duas câmeras se torna mais caro, há a necessidade de um passo extra de calibração e o consumo de recursos de processamento é maior. O emprego de apenas uma câmera é mais barato, facilita o desenvolvimento e é mais popular, visto que a maioria dos dispositivos móveis possui uma câmera. Não há necessidade de calibração e sincronização entre duas fontes de imagens. Na abordagem monocular a estimativa de profundidade não é possível com apenas uma imagem, é necessário a comparação entre vários quadros (AQEL et al., 2016). Neste caso a escala absoluta pode ser estimada empregando sensores inerciais, conhecendo a dimensão de outros objetos na cena ou a distância entre a câmera e o chão (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

2.5.5 Formulação do problema

Dado um fluxo de imagens capturado pela câmera de um dispositivo, em instantes adjacentes *k-1* e *k*, sendo representadas pelas imagens I_k e I_{K-1}. A concatenação das transformações relativas de $T_{k,\ k-1}$, relacionadas a transformação do corpo rígido, entre as duas imagens, são utilizadas para reconstituir a trajetória da câmera C. (SCARAMUZZA; FRAUNDORFER, 2011). A Figura 2 ilustra o paradigma da odometria visual, onde as transformações e posição da câmera são obtidas a partir da extração de características.

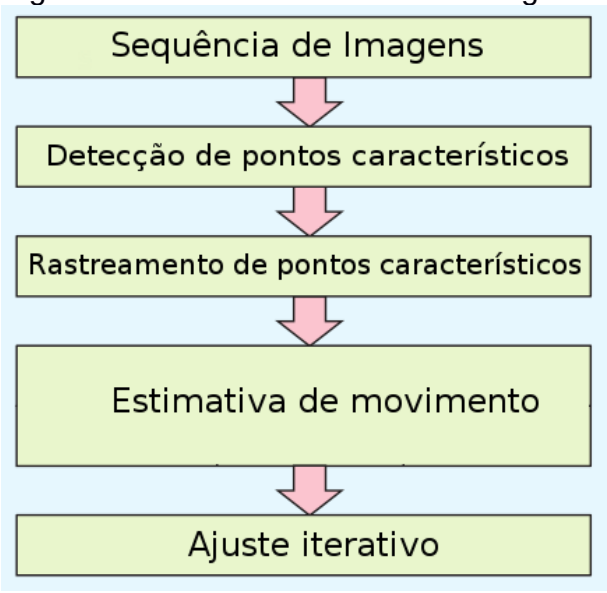
Figura 2 - Posições da câmera obtidas através da concatenação de transformações



Fonte: (SCARAMUZZA; FRAUNDORFER, 2011)

O algoritmo de odometria visual monocular, exibido na Figura 3, geralmente segue cinco passos. Inicialmente uma nova imagem I_k é obtida , na sequência é feita a detecção e a correspondência ou rastreamento das caraterísticas com a imagem anterior I_{k-1}. No terceiro passo é feito o cálculo da transformação relativa aos instantes *k-1* e *k* , o resultado é então utilizado para atualizar a posição da câmera no instante C_k. No estágio final, por questões de aperfeiçoamento, é possível realizar um ajuste iterativo a fim de melhorar a precisão da posição (SCARAMUZZA; FRAUNDORFER, 2011).

Figura 3 - Passos relacionados ao algoritmo de odometria visual monocular



Fonte: (SCARAMUZZA; FRAUNDORFER, 2011)

2.5.6 Extração de pontos característicos

Pontos característicos de uma imagem podem ser exemplificados como padrões salientes, que se diferenciam quanto a cor, intensidade e textura, em relação a suas proximidades (SCARAMUZZA; FRAUNDORFER, 2012). Os melhores pontos característicos são aqueles que apresentam facilidade de distinção em comparação com o ambiente. Geralmente estas características se apresentam como objetos, linhas, pontos, cantos, círculos ou polígonos (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2012). A maioria das pesquisas levam em consideração a detecção de cantos ou linhas. Os algoritmos que encontram estes padrões na imagem são chamados de detectores de características (SHELLEY, 2014).

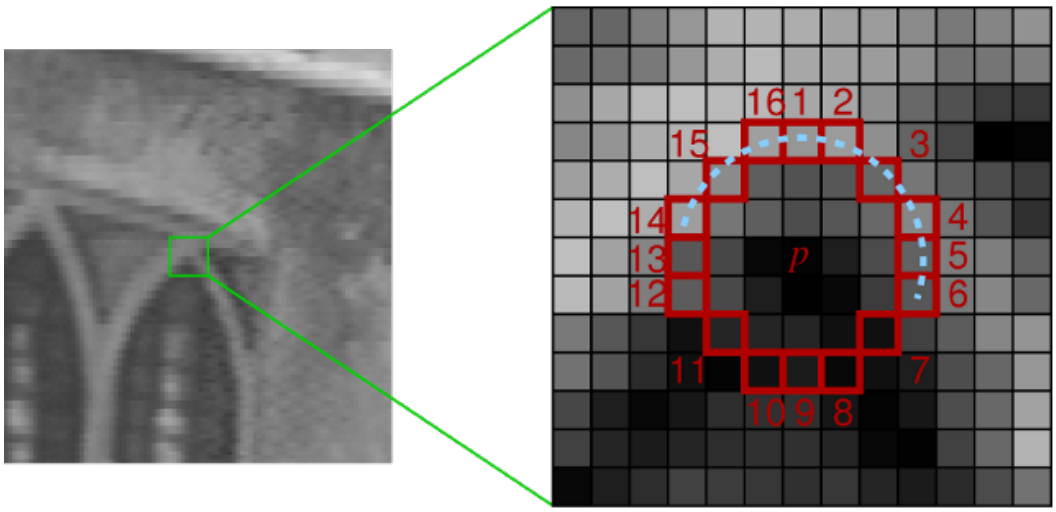
Um canto pode ser definido como a intersecção de dois contornos, onde há uma grande variação de intensidade dos *pixels* em todas as direções. Basicamente os algoritmos de extração de cantos devem proporcionar precisão na localização, quantidade de detecções suficientes para possibilitar a análise da correspondência entre imagens, invariância quanto a perspectiva da câmera, iluminação e mudança de escala. Além disso, a técnica precisa ser eficiente em termos de processamento e

apresentar robustez quanto a interferências e nitidez da imagem (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

O FAST (Features from Accelerated Segment Test) é um algoritmo de detecção muito utilizado em aplicações de odometria visual por sua velocidade e alta taxa de detecção. Seu funcionamento tem como princípio a comparação dentro de uma janela circular de 16 *pixels*, marcados em vermelho, conforme Figura 4. Se pelo menos 9 *pixels* contíguos da janela tiverem uma intensidade maior ou menor que um limite determinado, em relação ao centro da janela, o *pixel* central *p* é eleito como um possível canto (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016).

O detector FAST possui alguns pontos fracos, da mesma forma que outros algoritmos, ele depende de superfícies com textura e imagens com boa nitidez. Ambientes onde há apenas paredes brancas e corredores, assim como imagens desfocadas, dificultam a detecção de pontos (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016).

Figura 4 - Identificação da janela circular utilizada pelo algoritmo FAST



Fonte: Adaptado de (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016)

2.5.7 Correspondência ou rastreamento dos pontos característicos

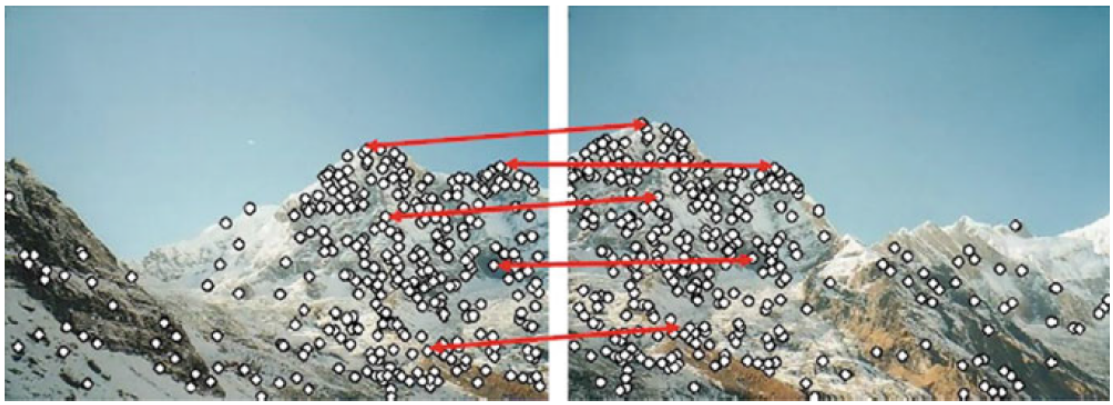
Existem basicamente duas técnicas para encontrar pontos característicos correspondentes entre as imagens. A primeira implica em encontrar pontos

característicos na imagem anterior, e depois pesquisar a ocorrência na imagem posterior. A segunda consiste em encontrar os pontos característicos em ambas imagens, e posteriormente empregar métricas de similaridade baseada em descritores (SCARAMUZZA; FRAUNDORFER, 2012). Um descritor é uma função aplicada a uma área da imagem, a fim de descrevê-la, de forma que o reconhecimento seja invariável a rotação, iluminação e ruído (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016).

Descritores binários, como o BRIEF (*Binary Robust Independent Elementary Features*), são gerados a partir da vizinhança do ponto característico, comparando a intensidade dos pixeis próximos, criando uma espécie de identidade binária única para cada ponto. O descritor ORB (Oriented FAST and Rotated BRIEF), baseado no BRIEF, acrescenta tolerância a rotação, sendo capaz de realizar a correspondência mesmo com mudanças de orientação da câmera (SHELLEY, 2014).

A comparação entre pontos similares é empregado em várias aplicações de computação visual. Após gerados os descritores de ambas imagens, as melhores correspondências são escolhidas, em termos de distância e similaridade (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016). A Figura 5 mostra a correspondência entre pontos descritos em diferentes imagens.

Figura 5 - Duas imagens distintas e seus descritores associados



Fonte: (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016)

Associações equivocadas ou ambiguidade entre pontos descritos, podem ocorrer. Algoritmos de extração de pontos característicos e descrição podem falhar ao serem aplicados em imagens distorcidas ou com baixa qualidade. Possíveis

causas para isso são ruídos nas imagens, imagens desfocadas, mudanças na iluminação e posição da câmera. Para definir as associações com mais assertividade, testes de consistência podem ser usados. Uma das táticas mais eficientes, consiste em pesquisar as potenciais correspondências nas áreas da imagem onde elas supostamente estariam. Ao invés de pesquisar todos os descritores encontrados na imagem, para descobrir a correspondência de um ponto (SCARAMUZZA; FRAUNDORFER, 2012).

Para que a movimentação da câmera possa ser estimada corretamente, é importante a remoção destes erros ou anomalias de associação. Para esta tarefa, no campo da odometria visual, o método RANSAC (*Random Sample Consensus*) vem sendo estabelecido como padrão. (SCARAMUZZA; FRAUNDORFER, 2012). O RANSAC é um método probabilístico e não determinístico, utilizado para remoção de anomalias. Sua função é identificar os pontos correspondentes, em imagens diferentes, que satisfaçam um modelo matemático pré-definido, como a movimentação seguindo o modelo de corpo rígido (HASSABALLAH; ABDELMGEID; ALSHAZLY, 2016).

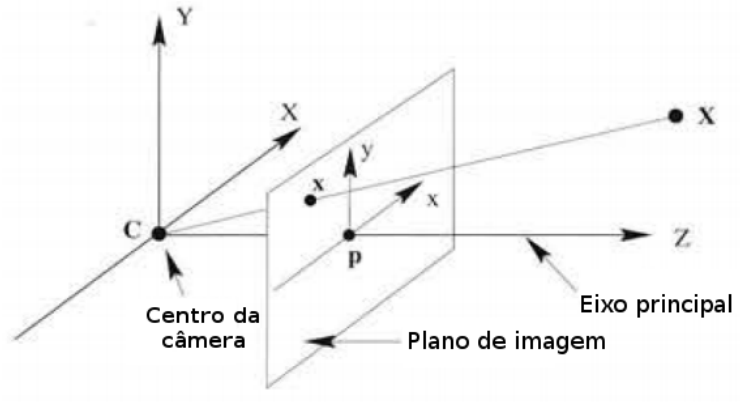
2.5.8 Projeção perspectiva

O modelo de câmera *pinhole*, ou câmera obscura, foi o primeiro exemplo de câmera, que levou a invenção da fotografia. O funcionamento é baseado em uma pequena abertura, feita em um dos lados de uma caixa, sem permitir a entrada de outra fonte de luz. Os raios de luz entram somente por um orifício central, fazendo uma projeção da imagem do lado oposto. É utilizada como base para demonstrar a projeção perspectiva dos pontos 3D vistos no mundo, em pontos 2D na imagem (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

As coordenadas X, Y, Z têm como ponto de referência o centro óptico C. Considerando C como origem, o plano $Z = f$, conhecido como plano da imagem ou plano focal, é posicionado em frente ao centro de projeção, a uma distância f, denominada distância focal. Perpendicular ao plano de imagem, a linha que parte o ponto C e intercepta o centro do plano de imagem p, é chamado eixo principal ou

eixo óptico. A Figura 6 demonstra um ponto *X = [X,Y,Z]* da cena projetado no ponto *x = [x, y]* do plano de imagem.

Figura 6 - Modelo de câmera pinhole



Fonte: Adaptado de (HARTLEY, ZISSERMAN, 2004)

A câmera *pinhole* modela uma equação para representação dos pontos no espaço, e o seu respectivo mapeamento na imagem. A formulação da equação, por motivos de simplificação, é representada através de matrizes. No formato de matriz, os pontos podem ser representados em coordenadas euclidianas ou através de coordenadas homogêneas.

A geometria projetiva é algebricamente representada pelo sistema de coordenadas homogêneas. Com a utilização de coordenadas homogêneas as transformações da imagem, rotação, escala e projeção se tornam simples multiplicações de matrizes (NIXON; AGUADO, 2012). De acordo com Hartley e Zissermann (2004), considerando que o centro de projeção está localizado na origem do sistema de coordenadas 3D e o eixo óptico é colinear ao eixo *Z*, como mostra a Figura 6, um ponto no espaço com coordenadas $(X, Y, Z)^T$ é mapeado para um ponto no plano da imagem $(u, v)^T$ por meio da seguinte equação:

$$(u, v)^T = (fX/Z, fY/Z)^T \quad (1)$$

Se os pontos no espaço 3D e os pontos no plano da imagem são representados por coordenadas homogêneas, a equação 1 pode ser escrita em notação matricial como:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (2)$$

em que $\lambda = Z$ é o fator de escala homogêneo.

2.5.9 Estimativa de movimento

A estimativa do movimento é o cerne dos algoritmos de odometria visual. Nele as alterações causadas pela movimentação da câmera são analisadas em cada imagem captada. A movimentação percebida é então concatenada, para formar a trajetória do agente. Portanto, o problema da estimativa de movimento pode ser expresso como a tentativa de obter a transformação T_K da câmera, entre duas imagens subsequentes I_{k-1} e I_k, computadas através da extração de pontos característicos f_{k-1} e f_k, em instantes *k-1* e *k* (SCARAMUZZA; FRAUNDORFER, 2012).

No trabalho pioneiro de Nistér et al. (2004) foi desenvolvido um sistema de odometria visual em tempo real, utilizando um aparato de câmera monocular ou estéreo. A posição da câmera era estimada através da projeção de pontos 3D da cena em pontos 2D da imagem. O algoritmo consistia em três fases, a detecção de pontos característicos, rastreamento de pontos característicos e estimativa de movimento.

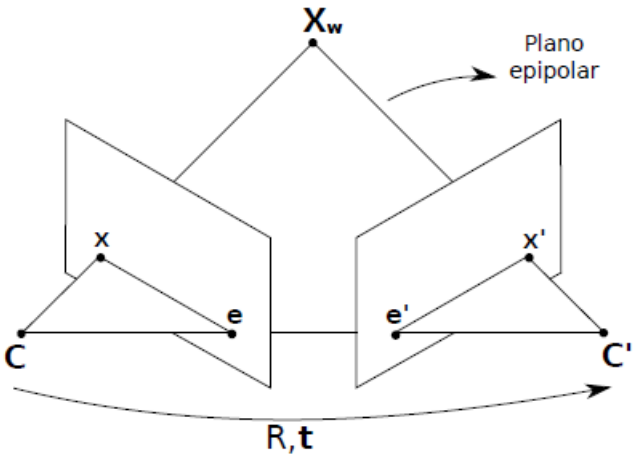
Não é possível recuperar informações de profundidade de pontos visualizados pela câmera utilizando somente uma imagem. Entretanto, coordenadas 3D de pontos característicos podem ser recuperados com informações presentes em mais de uma imagem. No caso monocular, o movimento pode ser estimado através de imagens capturadas em posições e orientações diferentes da câmera. A geometria que restringe pontos em duas imagens, chamada de geometria epipolar, pode ser utilizada para obter o movimento relativo da câmera como a estrutura 3D da cena visualizada (SCARAMUZZA; FRAUNDORFER, 2012).

Considerando duas imagens capturadas da mesma cena, em posições diferentes, cada par de imagem capturado por essa câmera representa duas perspectivas diferentes de uma mesma cena estática. A geometria epipolar estabelece uma relação geométrica entre duas vistas capturadas nessas condições. Tradicionalmente, esta geometria é empregada na busca por pontos correspondentes em algoritmos de visão estéreo (HARTLEY, ZISSERMAN, 2004).

Um ponto *X* no ambiente, as suas projeções na primeira e segunda imagem *x* e *x´*, tendo como centros ópticos das câmeras em *C* e *C´*. A união destes pontos forma o chamado plano epipolar. A Figura 7 exemplifica a formação do plano, e mostra também os epipolos *e* e *e´*, projeção dos centros ópticos nos planos de imagem, e as linas epipolares $\overline{ex}$ e $\overline{e´x´}$.

Em termos de algoritmos de visão estéreo destinados à computação de correspondências de pontos, o benefício proporcionado pela geometria epipolar é que, dado um ponto *x* referente à primeira vista, a busca por pontos correspondentes na segunda vista não precisa cobrir toda a imagem e restringe-se apenas a uma linha. Esta restrição também é conhecida na literatura como restrição epipolar (HARTLEY, ZISSERMAN, 2004).

Figura 7 - Formação do plano epipolar



Fonte: Adaptado de (HARTLEY, ZISSERMAN, 2004)

De acordo com Hartley, Zissermann (2004) a representação algébrica da geometria epipolar pode ser obtida a partir da matriz fundamental. A matriz fundamental representa um mapeamento projetivo de pontos de uma imagem para

linhas epipolares de outra. Segundo Siegwart, Nourbakhsh, Scaramuzza (2011) a partir da matriz essencial, derivada da matriz fundamental, é possível extrair os movimentos de rotação e translação da câmera. A matriz essencial pode ser computada utilizando o algoritmo de cinco pontos, utilizado por Nistér (2004). As coordenadas de escala, para inferir a movimentação para frente o para trás, podem ser obtidas através da triangulação de pontos correspondentes, em no mínimo três imagens subsequentes (SCARAMUZZA; FRAUNDORFER, 2012).

2.5.10 Homografia

De acordo com Dos Santos (2012) a homografia ‘”é uma transformação projetiva planar que mapeia pontos de um plano em outro”. Considerando duas imagens registradas de um diferente ponto de vista, a relação entre elas pode ser descrita matematicamente por uma matriz 3x3 chamada de matriz de homografia. Esta matriz, multiplicada pela representação homogênea de um ponto da imagem, resulta em outro ponto projetado no segundo plano (HARTLEY, ZISSERMAN, 2004). A Figura 8 mostra a equação onde um conjunto de pontos, um ponto *p = (x, y, w)* sobre um plano π é mapeado em um plano π’ , sobre o ponto correspondente *p’ = (x’, y’, w’)*.

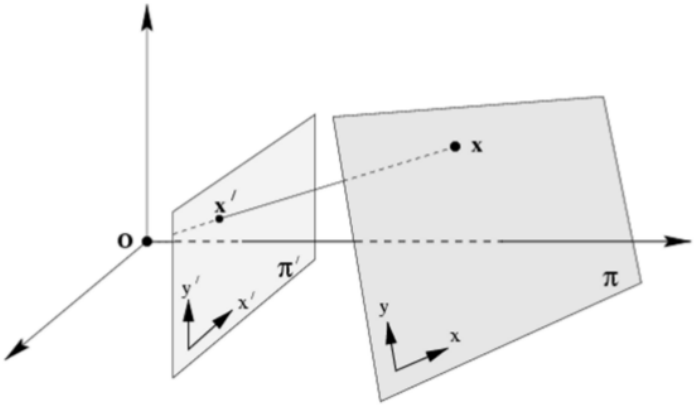
Figura 8 - Mapeamento de um ponto homográfico

$$\begin{bmatrix} x_i' \\ y_i' \\ w_i' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix}$$

Fonte: (HARTLEY, ZISSERMAN, 2004)

A Figura 9 exemplifica o mapeamento dos pontos em planos distintos. Traçando-se uma reta a partir da origem “O”, a linha intercepta os planos π e π' nos pontos *x’* e *x* .

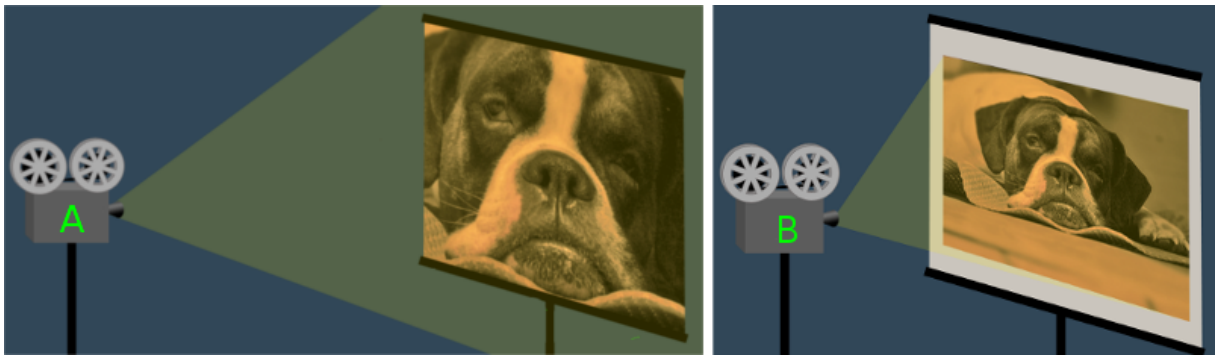
Figura 9 - Planos homográficos



Fonte: (HARTLEY, ZISSERMAN, 2004)

Esta relação permite que, com um mínimo de quatro pontos homográficos encontrados entre os planos, sejam estabelecidas as transformações projetivas entre as imagens. A Figura 10 mostra uma transformação homográfica de escala, o projetor A, mais afastado da tela de projeção, exibe a imagem em uma escala maior. Opostamente, o projetor B está mais perto da tela, mostrando a projeção em uma escala maior.

Figura 10 - Exemplo de transformação homográfica de escala



Fonte: Adaptado de (DALLING, 2017)

2.6 Localização baseada em imagens

Hoje em dia é comum a pesquisa de imagens informando alguma descrição textual ou metadados associados. Nos últimos anos o número de imagens produzidas e armazenadas, com a massificação da Internet e dispositivos móveis, vem crescendo consideravelmente. A demanda por maneiras mais inteligentes de

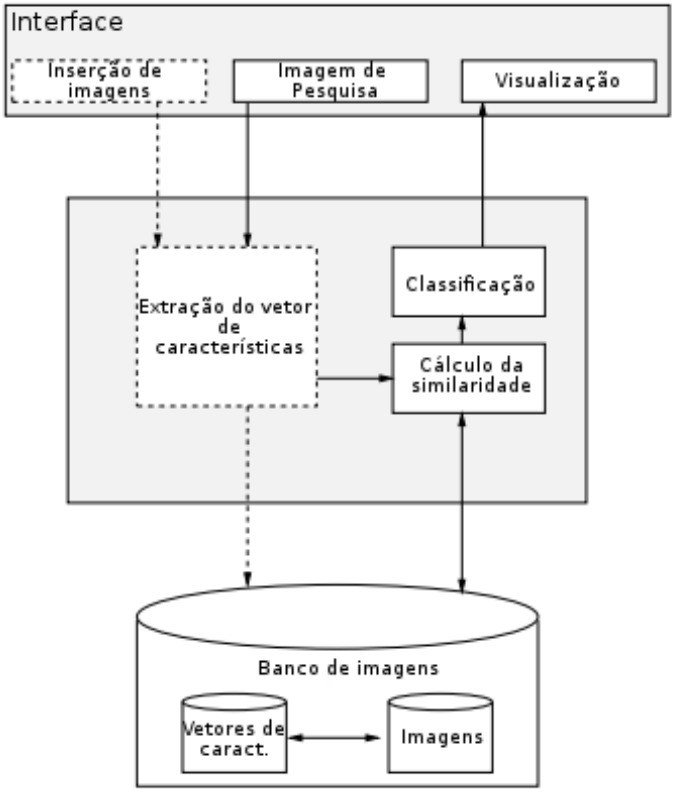
busca tem aumentado. Técnicas de busca em que a imagem possa ser pesquisada diretamente, utilizando as informações presentes nela mesmo, ao invés de suas descrições textuais, vem ganhando espaço. Neste campo vem se descantando o conceito de CBIR (Content-Based Image Retrieval), envolvendo as tecnologias que auxiliam a organização das imagens digitais, baseadas no seu conteúdo visual (BHARTI; WADHWA, 2013).

Para Santhi e Acharjya (2016) CBIR é definido como um conjunto de técnicas para recuperação de imagens de um banco dados considerando a extração características. Características como o conteúdo, histograma e disposição de cores, textura e formas, são utilizadas como parâmetro para distinção entre imagens.

Inicialmente, os sistemas CBIR aplicam algoritmos de processamento de imagem, onde são criados descritores para as características das imagens. Os descritores codificam as características e geram um vetor que representa abstratamente a imagem. Desta maneira o vetor pode ser comparado utilizando técnicas de similaridade. As funções de similaridade comparam os vetores e indicam a probabilidade de correspondência entre eles (SANTHI; ACHARJYA, 2016).

A Figura 11 mostra a arquitetura do sistema. Como entradas o CBIR deve proporcionar meios de cadastrar novas imagens e fazer a pesquisa. O núcleo do sistema é responsável por extrair o vetor de representação das imagens, pesquisar vetores semelhantes no banco de dados e estabelecer uma classificação dos resultados encontrados para exibição ao usuário. No banco de dados são armazenados somente os vetores de representação das imagens.

Figura 11 - Arquitetura de um sistema CBIR



Fonte: Adaptado de (TORRES; FALCÃO, 2006)

2.7 Desenvolvimento mobile híbrido

O desenvolvimento híbrido surgiu para facilitar a construção de aplicativos mobile multiplataforma. Um aplicativo híbrido é codificado utilizando linguagens web como HTML, CSS e JavaScript ao invés da programação tradicional, utilizando a linguagem nativa do dispositivo. Com isso é possível utilizar o código em diversas plataformas, reduzindo o tempo de desenvolvimento (LOPES, 2016). Diversos são os desafios encontrados para criação de um aplicativo, cada plataforma tem filosofia de desenvolvimento independente e maneiras distintas de lidar com problemas. Desenvolver, testar e manter uma aplicação nativa tende a ser cara (SALEH et al., 2016).

Quando um aplicativo híbrido é executado, um componente chamado WebView é utilizado para exibir o conteúdo da página HTML na janela da aplicação.

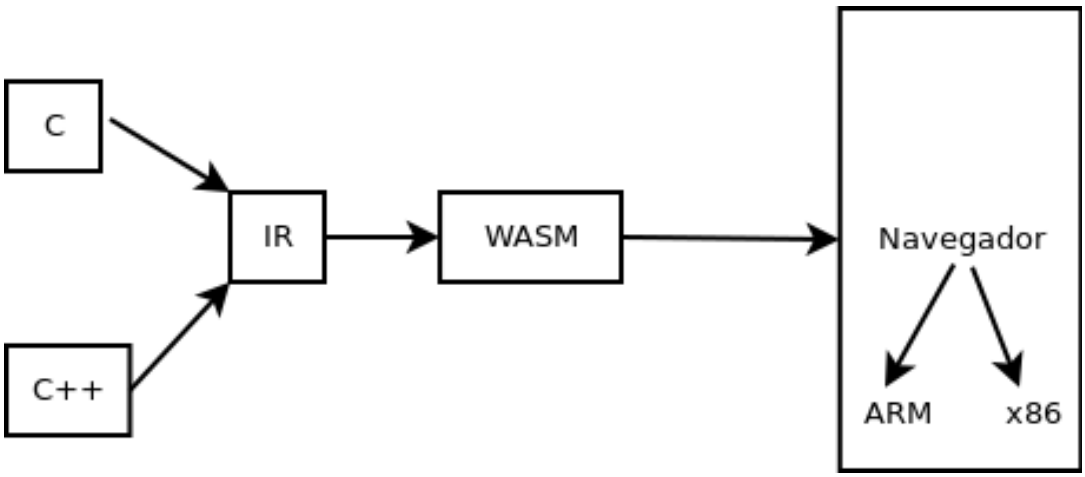
O WebView funciona como um *browser* que é executado dentro da aplicação (SALEH et al., 2016). Segundo Lopes (2016), do ponto de vista técnico, para o usuário final não diferença entre a aplicação nativa e híbrida. O aplicativo híbrido, se bem construído, integra-se a plataforma da mesma forma que o nativo. Aplicativos nativos são mais utilizados quando há a necessidade de extrair o máximo do hardware, caso dos jogos e aplicações que necessitem de *multithreading*.

2.8 WebAssembly

O WebAssembly ou wasm é um novo formato para representação de código baixo nível, muito parecido com Assembly, que pode ser executado em *browsers* modernos. Seu principal objetivo é permitir que outras linguagens possam ser empregadas para o desenvolvimento *web*, além do JavaScript. Criada para ser um alvo de compilação para linguagens como C e C++, programas escritos nestas linguagens podem ser simplesmente traduzidos e executados em ambiente *web*. Por ser um formato binário e compacto, sua execução e performance assemelha-se ao código de máquina nativo. Desta maneira, o ganho de desempenho acontece por que o navegador consegue decodificar e executar as instruções binárias muito mais rápido que o JavaScript (MOZILLA, 2017c).

A Figura 12 detalha o fluxo de desenvolvimento, primeiramente o código fonte em C ou C++ é compilado para uma representação intermediária (RI) comum. O código intermediário é então traduzido para wasm. O wasm não é a representação das instruções para máquina física, mas sim instruções para uma máquina conceitual, implementada pelo próprio motor JavaScript, que faz a tradução para a arquitetura da máquina do usuário (CLARK, 2017b).

Figura 12 - Estrutura de desenvolvimento WebAssembly



Fonte: Do autor (2017)

De acordo com o site oficial, o WebAssembly é um novo formato, portável, leve e com tempo de carregamento eficiente, adequado para compilação na Web. Assim como o formato JavaScript, com extensão “.js”, o WebAssembly utiliza o “.wasm”. Foi desenhado para rodar eficientemente em diferentes sistemas operacionais e arquiteturas. Seu tempo de carregamento tende a ser mais rápido, já que assume uma representação binária com tamanho de arquivo menor que o JavaScript. Apesar de todas estas melhorias, o WebAssembly não pretende ser uma substituição ao JavaScript, mas ser um complemento, preenchendo os espaços, onde é necessário uma maior eficiência no processamento no lado do cliente (WEBASSEMBLY, 2017).

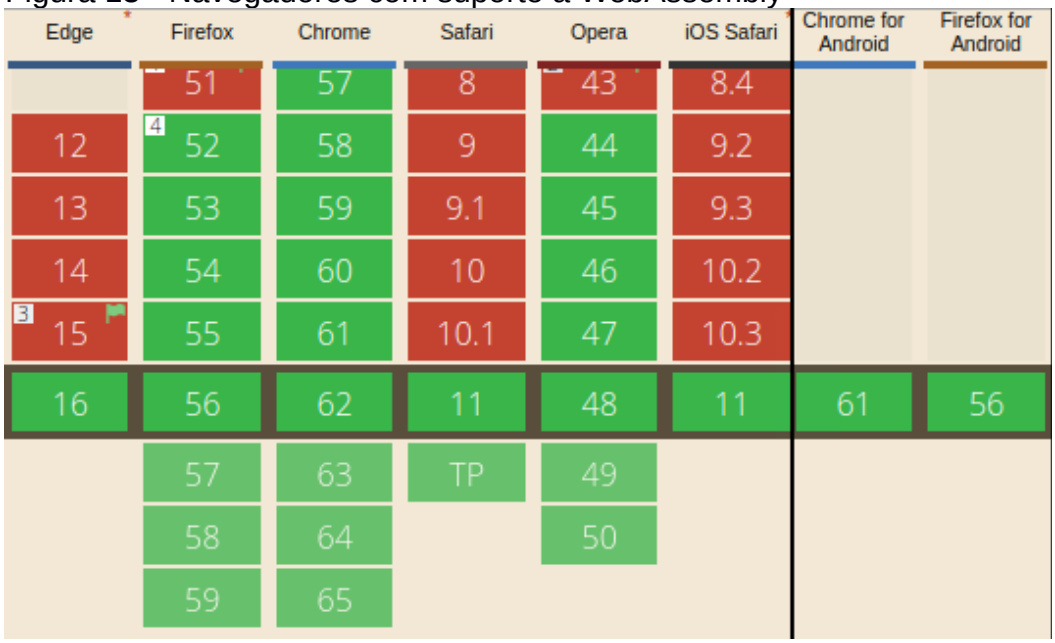
Recentemente, mais precisamente no final de fevereiro de 2017, os quatro maiores fabricantes de *browsers*, anunciaram um consenso sobre o WebAssembly. Os desenvolvedores do Chrome, Edge, Firefox e WebKit definiram as especificações iniciais, ou MVP (Minimum Viable Product), com as características e objetivos básicos do novo formato (CLARK, 2017a). Isto define uma versão zero ou base, para que o WebAssembly possa se tornar utilizável nos navegadores. O WebAssembly está sendo desenvolvido como um padrão aberto, por um grupo da comunidade W3C. Seus principais objetivos são (MOZZILLA, 2017b):

- Rapidez, eficiência e portabilidade: possibilidade de ser executado a uma velocidade praticamente nativa por diferentes plataformas;

- Legibilidade e de fácil análise: apesar de ser um formato binário, possuir uma representação que possa ser escrita e depurada;
- Segurança: execução em um ambiente separado, com as mesmas restrições de origem e permissões do navegador;
- Integração: ser desenvolvida de maneira que interação com outras tecnologias *web* e retrocompatibilidade seja mantida.

Os navegadores mais populares já contam com suporte ao WebAssembly, de acordo com o site Caniuse (CAN, 2017) o WebAssembly já está presente em 61% dos navegadores utilizados atualmente. O site mostra também um comparativo, mostrando um histórico das versões suportados. A Figura 13 relaciona os principais navegadores, em verde estão marcadas as versões habilitadas com WebAssembly, em vermelho aparecem as versões que não oferecem suporte. O Firefox por exemplo, oferece suporte desde a sua versão 52 lançada em março de 2017. O Chrome por sua vez, desde a versão 57 lançada em 8 de março de 2017.

Figura 13 - Navegadores com suporte a WebAssembly



Fonte: (CAN, 2017)

2.9 Algoritmo A-star

O algoritmo A* ou A estrela, é provavelmente um dos algoritmos mais utilizados para encontrar o mínimo caminho em mapas, amplamente empregado no desenvolvimento de jogos. É uma evolução do algoritmo de Dijkstra, adicionando uma etapa de decisão heurística para escolher o melhor nodo para avançar. No algoritmo de Dijkstra, a cada iteração são analisados todos os nodos adjacentes, de modo que toda vizinhança é explorada, e o avanço é baseado na soma dos custos. Entretanto, o algoritmo A* pesquisa o nodo baseado, por exemplo, na distância euclideana, em relação ao destino. Com isso o algoritmo explora os nodos avançando em direção ao destino final (CELES; CERQUEIRA; RANGEL, 2016).

3 TRABALHOS RELACIONADOS

Este capítulo lista os estudos que aplicam conceitos semelhantes aos empregados neste trabalho. Muitas são as pesquisas relacionadas a odometria visual e reconhecimento de imagens para localização *indoor*.

3.1 HyMoTrack

Gerstweiler, Vonach e Kaufmann (2015) desenvolveram um sistema chamado HyMoTrack, para navegação *indoor*, utilizando dispositivos móveis. O algoritmo emprega sensores inerciais, câmera e marcos visuais do ambiente para localização, todo o processamento é feito localmente no dispositivo. No primeiro estágio, existe um mapeamento do ambiente, um computador e uma câmera são utilizados para gravar informações espaciais, com ajuda de um algoritmo de SLAM (Simultaneous localization and mapping). Nesta etapa os marcos visuais 2D são registrados para auxiliar na obtenção das coordenadas.

No segundo estágio de navegação foi desenvolvido um aplicativo Android. Nesta etapa as imagens do ambiente são capturadas e comparadas com as informações visuais adquiridos na fase de mapeamento. O usuário ao deslocar-se no ambiente *indoor*, com a câmera do dispositivo captando imagens do espaço, pode visualizar setas e placas indicativas, sobrepostas na tela do dispositivo, com auxílio de realidade aumentada. Se houver alguma falha na navegação visual, os dados dos sensores inerciais são utilizados.

Os testes foram realizados no Aeroporto Internacional de Viena, em uma área onde costumeiramente os passageiros tem dúvidas sobre sua orientação. A Figura 14 exemplifica o uso da aplicação. Uma pessoa se desloca pelo aeroporto e vê na tela do *tablet* as indicações de direção, exibidas com auxílio de realidade aumentada.

Figura 14 - HyMoTrack utilizado no Aeroporto de Viena

Fonte: Adaptado de (GERSTWEILER, VONACH e KAUFMANN, 2015)

3.2 Localização baseada em imagem para ambientes *indoor* utilizando dispositivo móvel

Huang et al. (2015) propõe um sistema onde o usuário pode se localizar registrando apenas uma imagem do seu entorno. O resultado é alcançado utilizando correspondência e técnicas de pesquisa em um banco de imagens pré capturadas e referenciadas com coordenadas. Todo o processo ocorre no dispositivo, vários algoritmos para a análise das texturas e detecção de pontos característicos foram testados.

No experimento proposto, várias imagens de um trajeto *indoor* são registradas perpendicularmente, com uma sobrexposição de mínimo de 70% e iluminação adequada. A Figura 15 expõe a sequência de imagens capturadas de um estande de exibição, cada uma foi registrada a partir de uma posição diferente.

Figura 15 - Conjunto de imagens e posição de captura

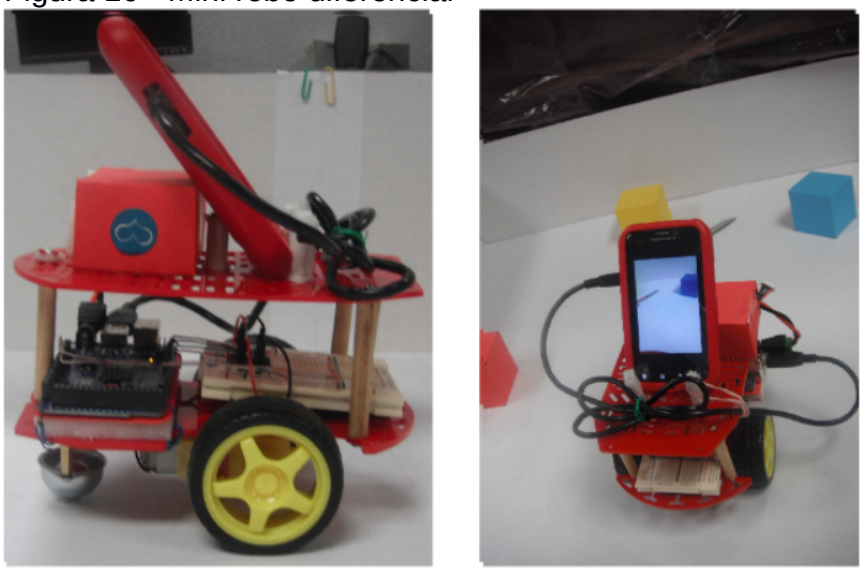
Fonte: Adaptado de (HUANG ET. AL, 2015)

As imagens vizinhas são alinhadas a partir da extração e rastreio de pontos característicos. Isto permitiu estimar a posição relativa da câmera para cada par de imagem. Os pontos das imagens e as posições da câmera são inseridas em um banco de dados estruturado, para possibilitar a pesquisa de posição. Na etapa de localização, o usuário é capaz de obter o seu local atual com um registro de uma imagem da cena *indoor*.

3.3 Odometria visual monocular para robôs

No trabalho de Villanueva-Escudero et al. (2014) um sistema de odometria visual monocular é proposto. A Figura 16 mostra o sistema construído, que utiliza um *smartphone* Android para controlar um pequeno robô com duas rodas e estimar a odometria visual. A partir do fluxo de imagens adquiridos pela câmera do *smartphone*, são extraídos pontos característicos com o algoritmo FAST. O cálculo de movimentação tem como base o algoritmo de cinco pontos, proposto por Nistér (2004). O experimento utilizou o SDK OpenCV para Android.

Figura 16 - Mini robô diferencial



Fonte: (VILLANUEVA-ESCUDERO ET AL. ,2014)

4 METODOLOGIA

Neste capítulo serão contextualizados os processos e ferramentas para desenvolvimento do trabalho, bem como uma breve introdução sobre o tipo de pesquisa relativa ao projeto.

Considerando áreas envolvidas, esta pesquisa pode ser considerada exploratória, de modo que pretende-se verificar e conhecer o funcionamento das ferramentas e teorias abordadas. Segundo GIL (2008, p. 27):

> As pesquisas exploratórias têm como principal finalidade desenvolver, esclarecer e modificar conceitos e ideias, tendo em vista a formulação de problemas mais precisos ou hipóteses pesquisáveis para estudos posteriores.

Paralelamente, o estudo apresentado tem o caráter experimental. GIL (2008, p. 51) define que:

> o delineamento experimental consiste em determinar um objeto de estudo, selecionar as variáveis que seriam capazes de influenciá-lo, definir as formas de controle e de observação dos efeitos que a variável produz no objeto.

4.1 Organização da Pesquisa

Inicialmente foi analisada a viabilidade de implementação da odometria visual, utilizando apenas o caso monocular. Detectou-se a possibilidade técnica, visto a quantidade significativa de trabalhos relatando sucesso na execução. Em seguida,

procurou-se uma ferramenta ou biblioteca que disponibilizasse os algoritmos e recursos necessários a construção de um aplicativo para *smartphones* Android. A grande maioria das implementações utiliza a biblioteca de visão computacional OpenCV (OPENCV, 2017a). Originalmente escrita com a linguagem de programação C++, esta biblioteca disponibiliza um SDK (*Software Development Kit*) para plataforma Android.

Neste momento, buscou-se encontrar alternativas que facilitassem o desenvolvimento. Foram encontradas bibliotecas JavaScript voltadas a computação visual. Estas bibliotecas disponibilizam vários recursos necessários a implementação da odometria, além de facilitar a instalação, visto que são executadas em um ambiente *web*, necessitando apenas um navegador. Sendo assim, decidiu-se desenvolver o aplicativo para Android em uma plataforma híbrida, utilizando apenas HTML, CSS e JavaScript.

No entanto, após alguns testes, foi detectado que as bibliotecas JSFeat e Tracking.JS não possuíam um desempenho satisfatório para o processamento da odometria visual. O processamento do algoritmo, a análise dos quadros da imagem, utilizando JavaScript mostrou uma baixa taxa de FPS (Frames Per Second), o que inviabilizou o seu uso. Neste momento surgiu a possibilidade do uso do WebAssembly, com a finalidade de trazer maior performance e possibilitar o uso da funções da biblioteca OpenCV, proporcionando a utilização dos seus recursos e algoritmos de visão computacional.

O próximo passo foi propor uma emprego prático da odometria visual. Surgiu então a ideia de utilizá-la em um aplicativo de navegação *indoor* no *smartphone*. Com isso os usuários podem acompanhar em tempo real o seu deslocamento dentro do ambiente. Os últimos desafios encontrados foram o desenvolvimento do sistema de *wayfinding* e a forma de inferir a localização inicial, visto que o único sensor utilizado é a câmera. A solução para o sistema de *wayfinding* foi uso de uma biblioteca JavaScript que implementa o algoritmo de busca A* (A *star*). Para a inferir localização inicial, um sistema CBIR fez o reconhecimento da posição a partir de uma imagem registrada pelo aplicativo.

4.2 Ferramentas

Esta seção lista as ferramentas envolvidas no trabalho. A *interface* do aplicativo foi codificada com HTML, JavaScript e CSS. O Framework7 auxiliou na estruturação da aplicação, através da padronização de componentes e estilos de renderização. Para implementar o mapa e a pesquisa do melhor trajeto, entre os pontos de origem e destino, foi utilizada a biblioteca JavaScript javascript-astar. O *framework* Cordova foi o responsável pela construção do aplicativo híbrido.

No processo de localização inicial, a seleção do ponto de origem empregou o CBIRest. O teste preliminar, para estimativa de movimento, foi realizado com as bibliotecas JSFeat e TrackingJS. Na segunda experimentação, utilizando a biblioteca OpenCV, as ferramentas CMake, Ninja e Emscripten foram utilizadas para complilação do projeto C++ e geração do ".wasm".

4.2.4 Framework Cordova

O Apache Cordova é um *framework* de código aberto, para desenvolvimento *mobile* híbrido. O papel do Cordova é encapsular o código *web* e executá-lo dentro de um navegador, bem como permitir o acesso a recursos nativos do dispositivo. Ele fornece uma maneira de desenvolver aplicações mobile independentes da plataforma. Disponibilizando todas as ferramentas para empacotamento do código em aplicativo nativo, capaz de abrir uma W*ebView* e executar o código HTML, CSS e JavaScript. Os recursos nativos do dispositivo são acessíveis por meios de *plugins.* O *framework* Cordova disponibiliza um grande repositório, onde é possível encontrar *plugins* para, por exemplo, manipular os sensores, câmera, lista de contatos, dentro outros (APACHE, 2017).

Antes de ser Apache Cordova, o *framework* chamava-se PhoneGap (PHONEGAP, 2017). O PhoneGap foi criado em 2009 pela empresa Nitobi. Em 2011, a empresa Adobe comprou a empresa mas doou todo o código para o projeto Apache. A versão atual, estável, utilizada neste projeto é a 7.0.

4.2.5 Bibliotecas JavaScript

O JavaScript é uma linguagem de programação criada para navegadores, com o objetivo de proporcionar maior interatividade às páginas web, é suportada hoje por todos os navegadores (MOZILLA, 2017a). Em um *ranking* elaborado pelo GitHub (GITHUB, 2017a) mostra a liderança do JavaScript em número de repositórios abertos, sugerindo a preferência dos desenvolvedores pela linguagem. Atualmente o JavaScript é linguagem mais popular, de acordo com artigo publicado em janeiro de 2017, através de um cruzamento de dados de popularidade do *site* Stack Overflow (STACKOVERFLOW, 2017) e do GitHub (REDMONK, 2017).

Com o poder de processamento dos dispositivos móveis e computadores pessoais crescendo consideravelmente, a possibilidade de executar algoritmos de visão computacional, no lado do cliente, tornou-se viável. Muitas aplicações complexas podem agora ser realizadas no navegador de Internet, sem a necessidade de instalações extras (AKHMADEEV**,** 2015).

4.2.5.1 Biblioteca JSFeat

A biblioteca JavaScript JSFeat (INSPIRIT, 2017) reúne alguns dos algoritmos mais avançados utilizados para visão computacional. Funcionalidades como funções matemáticas, processamento de imagem, detecção e descrição de pontos característicos, e fluxo óptico estão presentes (AKHMADEEV**,** 2015). Esta biblioteca foi criada por apenas um desenvolvedor russo, chamado Eugene Zatepyakin. Seu repositório no GitHub mostra que o desenvolvimento iniciou por volta de 2012. A última atualização do repositório constatada foi em dezembro de 2015.

4.2.5.2 Biblioteca Tracking.JS

Assim como a JSFeat, a biblioteca Tracking.JS (TRACKINGJS, 2017) é voltada para visão computacional, em especial a detecção de faces, cores e objetos.

4.2.5.3 Biblioteca javascript-astar

O mapa de navegação será representado através de uma matriz de bi-dimensional. A biblioteca *javascript-astar* (GRINSTEAD, 2017) fornece um meio de exibir um labirinto baseando-se em uma matriz, onde o algarismo 0 representa um obstáculo e o 1 um caminho livre. Além disso, ela implementa o algoritmo de busca A*, a partir de um ponto de origem e destino, ela busca e exibe visualmente o caminho encontrado.

4.2.6 Aplicação CBIRest

Para a tarefa de reconhecimento de imagem será empregado o software CBIRest (CBIREST, 2017), *opensource* e desenvolvido em Java. Sua forma amigável de implantação, sua API (*Application Programming Interface*) REST (*Representational State Transfer*) para manipulação das informações e a possibilidade de acrescentar metadados foram alguns dos motivos para sua utilização neste projeto. Através de requisições HTTP são possíveis o cadastro de novas imagens e posterior pesquisa baseada no conteúdo da imagem. Além disso o CBIRest fornece também uma aplicação web para administração e manipulação das imagens.

4.2.4 OpenCV

O OpenCV é uma biblioteca de computação visual e aprendizado de máquina *open source,* escrita em C e C++. Lançada em 1999, com suporte inicial da Intel, um dos principais objetivos é prover uma infraestrutura simples e sólida para auxiliar no desenvolvimento de aplicações relacionadas a visão computacional. Seu uso abrange áreas como reconhecimento de face humana e classificação de ações, identificação de objetos, rastreamento de movimento e processamento de imagens. Atualmente conta com uma comunidade com mais de quarenta e sete mil usuários e apoio de grandes empresas como Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota (OPENCV, 2017b).

4.2.5 Framework7

A interface da aplicação foi confeccionada com o auxílio *framework* HTML Framework7. Ele facilita e agiliza o desenvolvimento de aplicativos híbridos, fornecendo uma infinidade de componentes pré-prontos, integrando-se perfeitamente ao visual das plataformas iOS e Android.

4.2.6 CMake, Ninja e Emscripten

O CMake é um sistema que gerencia o processo de *build*, compilação e linkedição de uma aplicação, para que esta possa se tornar executável. É um projeto *open source*, desenhado para ser utilizado independentemente do sistema operacional ou compilador (CMAKE, 2017). Diferentemente de outros sistemas, o CMake não executa o processo de *build,* mas sim traduz uma descrição de *build* alto nível em um formato baixo nível aceito por outras ferramentas. No sistema Linux, por exemplo, o CMake pode traduzir as definições em um makefile, pronto para uso da ferramenta GNU Make (SMITH, 2011).

O projeto Ninja (NINJA, 2017) dedica-se a criar um sistema de *build* com foco em desempenho. Projetado para receber como entrada arquivos gerados por ferramentas alto nível, como o CMake.

Emscripten (EMSCRIPTEN, 2017) é um compilador de bytecode LLVM (LLVM, 2017) para JavaScript. Qualquer linguagem que possa traduzida para LLVM pode ser portada para JavaScript. Por padrão o Emscriptem gera código compatível com asm.js, o suporte a WebAssembly pode ser adicionado a partir da instalação do compilador Binaryen (BINARYEN, 2017).

4.3 Ambiente de desenvolvimento

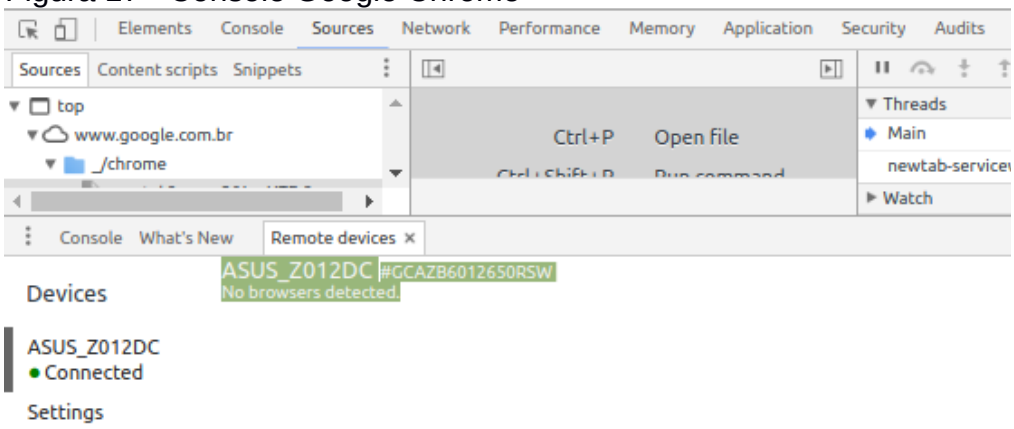
O desenvolvimento foi realizado em um Notebook Lenovo S400, processador CoreI5, 4GB de memória RAM (Random Access Memory) e armazenamento SSD (Solid State Drive). Neste equipamento, com sistema operacional Linux Mint 16.04,

foram instalados o framework de desenvolvimento Cordova, a aplicação CBIRest e a biblioteca OpenCV. Foi necessário também a instalação do Java SDK, Android SDK e Gradle, requisitos para funcionamento do Cordova.

Para os testes, foi utilizado o smartphone ASUS Zenfone 3, processador Qualcomm Snapdragon 625 Cortex A53, com 8 núcleos. O aparelho possui 4GB de memória RAM, armazenamento de 64GB e sistema operacional Android 7.0 Nougat. No experimento foi utilizada a câmera traseira de 16 Megapixel.

Para codificação foi utilizado a IDE (Integrated Development Environment) Visual Studio Code, ela possui suporte a todas as linguagens utilizadas neste projeto, através da adição de *plugins*. Na parte de *debug* foi utilizado basicamente o console do Google Chrome através da conexão com dispositivos remotos. A Figura 17 mostra as ferramentas para desenvolvedor do Chrome e a possibilidade de conexão a dispositivos remotos.

Figura 17 - Console Google Chrome



Fonte: Do autor (2017)

5 ATIVIDADES

Neste capítulo serão detalhadas as atividades relacionadas ao desenvolvimento do aplicativo e o seu funcionamento. O processo foi dividido em fases distintas, inicialmente foi realizada a construção do sistema de *wayfinding* e exibição do mapa 2D, seguindo da pesquisa baseada em imagem com o CBIRest. Por fim foram realizados os ensaios com a odometria visual. O resultado do trabalho foi dividido em dois repositórios distintos do GitHub, um denominado AppTcc (GITHUB, 2017b) e outro chamado AppTcc-wasm (GITHUB, 2017c). O primeiro contém o código do aplicativo híbrido em si, que é instalado no *smatphone*. No segundo está o projeto C++ que gera o ".wasm", utilizando as funções da biblioteca OpenCV, que faz o processamento da odometria visual.

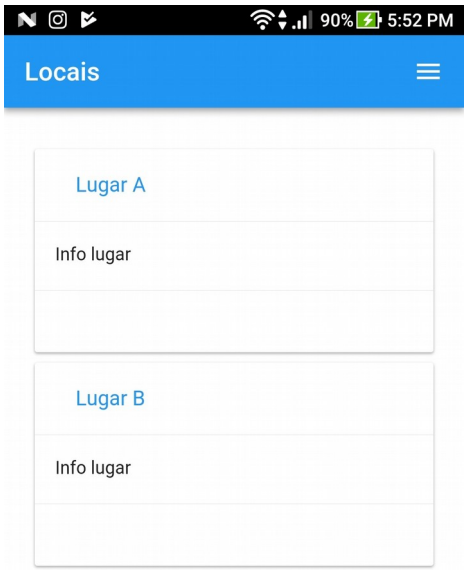
O experimento evoluiu até os testes com a odometria visual, onde foi possível captar o fluxo de imagem da câmera e fazer o processamento via WebAssembly. No entanto, não foi viável a integração desta parte com a exibição no mapa, visto que não foi possível atingir um nível de acuracidade nas estimativas de movimento. Nas seções seguintes são descritos os detalhes da implementação.

5.1 Desenvolvimento do aplicativo *mobile*

O experimento possui um cadastro prévio dos locais, registrados estaticamente na própria aplicação *mobile*, juntamente às suas coordenadas X e Y, com referência a um mapa 2D. O mapa 2D é uma representação simulada de um ambiente, onde a parte branca representa uma área livre para deslocamento e a parte preta um obstáculo. A Figura 18 mostra a lista de locais previamente cadastrados. Neste protótipo não é permitido ao usuário cadastrar novos locais, é possível apenas a inclusão de imagens relacionadas ao local.

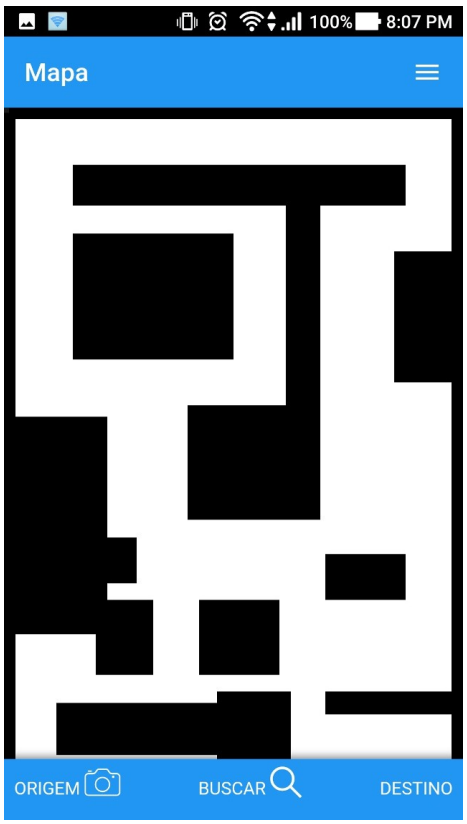
Na Figura 19 é apresentado o mapa de um ambiente simulado, a parte escura equivale as paredes e a parte branca os corredores.

Figura 18 - Lista de locais



Fonte: Do autor (2017)

Figura 19 - Mapa 2D

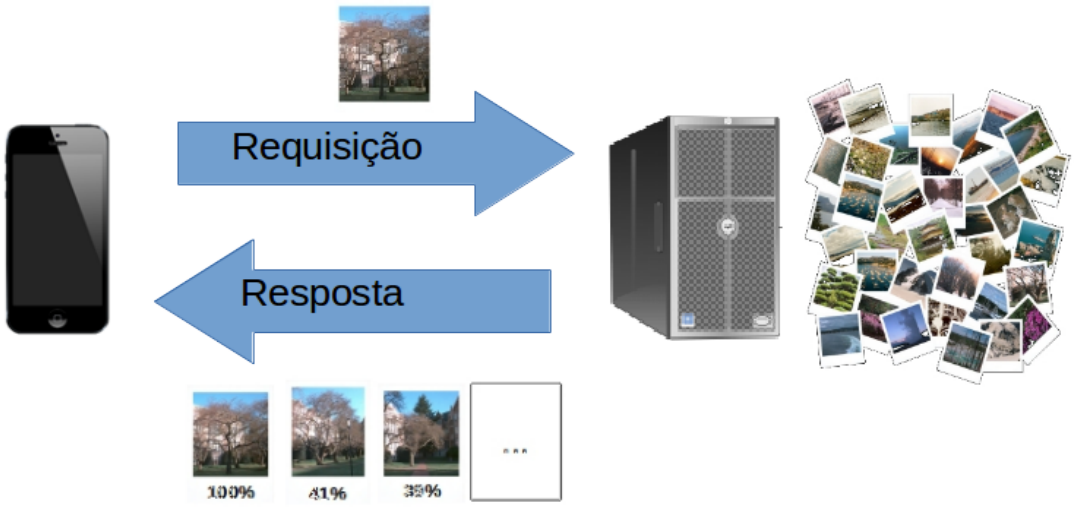


Fonte: Do autor (2017)

Após esta primeira fase, com relacionamento entre imagem e local, os usuários podem efetuar pesquisa com base na imagem registrada do ambiente. Com isso, ao clicar no botão “Origem”, mostrado na Figura 19, a interface da câmera abre, possibilitando a captura de uma imagem. Esta imagem é utilizado como parâmetro de pequisa.

O processo de cadastro de imagens e pesquisa acontecem no servidor CBIRest. A Figura 20 exemplifica o processo de pesquisa, onde o aplicativo envia a imagem ao servidor CBIRest, via rede. O CBIRest faz a busca, com base nas imagens cadastradas, retornado o registro mais similar, juntamente à coordenadas X e Y do local. Ao receber as coordenadas, esta posição é exibida no mapa, na indicação de um ponto verde, conforme Figura 21.

Figura 20 - Arquitetura da pesquisa CBIRest



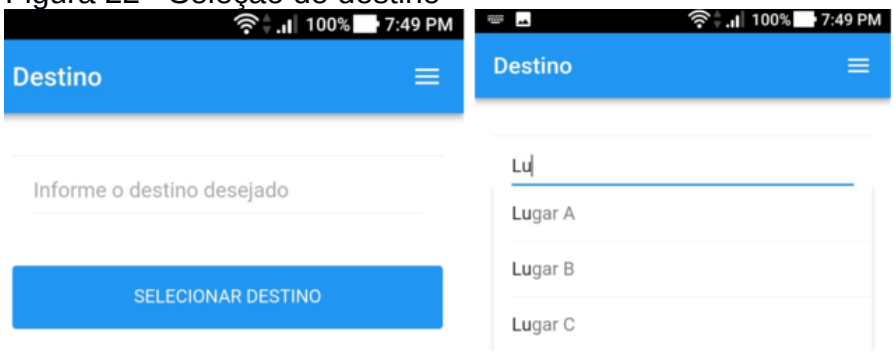
Fonte: Do autor (2017)

Figura 21 - Ponto de origem exibido no mapa

Fonte: Do autor (2017)

Após definir a origem, o usuário tem a possibilidade de selecionar um destino, clicando no botão “Destino”, exibido na Figura 19. A Figura 22 mostra a tela onde é feita a seleção do ponto de destino, através de uma pesquisa textual pelo nome do local.

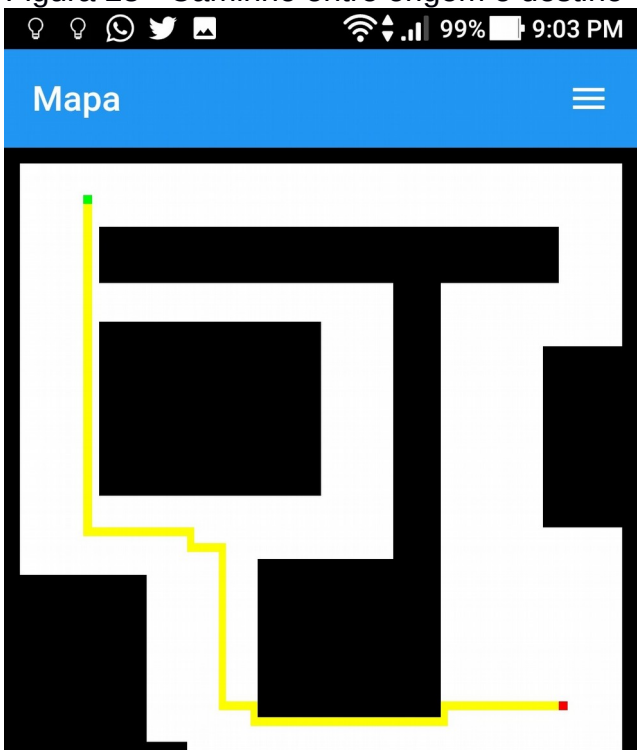
Figura 22 - Seleção de destino



Fonte: Do autor (2017)

Estabelecida a origem e o destino, o usuário pode então solicitar que o aplicativo trace a melhor caminho entre os dois pontos. Clicando no botão “Buscar”, que aparece na Figura 16, uma rota é traçada entre estes dois pontos. A pesquisa do melhor caminho é feita utilizando o algoritmo A* (A *star*). A Figura 23 apresenta, na cor amarela, a rota entre a origem e o destino.

Figura 23 - Caminho entre origem e destino



Fonte: Do autor (2017)

5.2 Integração com aplicação CBIRest

A comunicação entre o aplicativo e a aplicação CBIRest é feita através de requisições HTTP. O CBIRest fornece uma API para manipulação das imagens, o principal desafio então foi implementar o lado do cliente, no aplicativo. Foram necessários a instalação de alguns *plugins* do Cordova para possibilitar a comunicação:

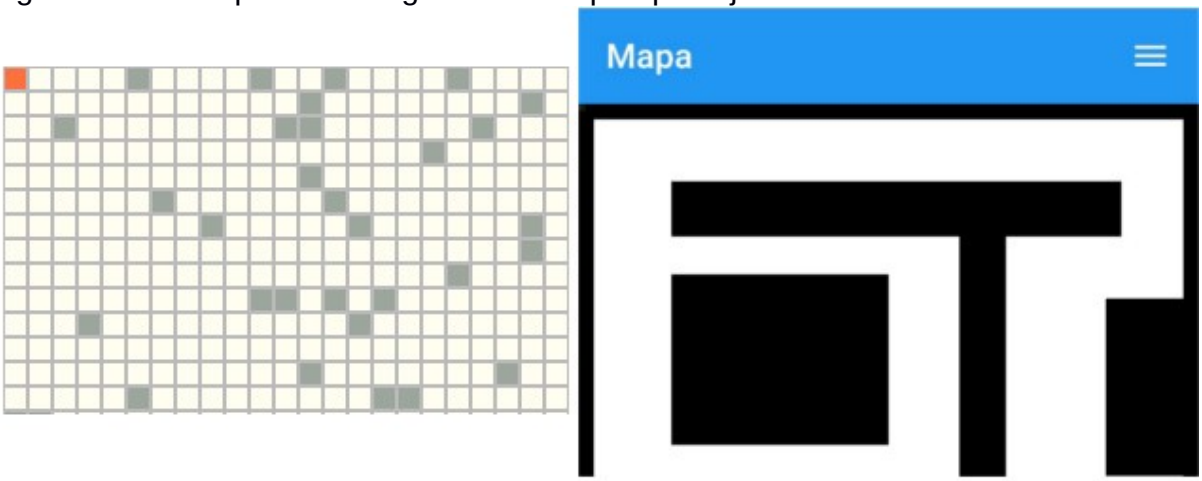
- cordova-plugin-http: habilita comunicação utilizando protocolo HTTP;
- cordova-plugin-file-transfer: proporciona o *upload* e *download* de arquivos através das requisições HTTP POST e PUT;
- cordova-plugin-camera: oferece uma API para registrar fotos utilizando a câmera.

A instalação da aplicação CBIRest pode ser encontrada no site do desenvolvedor (CBIREST, 2017). Consiste basicamente em baixar a aplicação web Java, extensão “.war”, e executá-la utilizando comando “java -jar”.

5.3 Criação do mapa 2D e exibição do melhor caminho

Para criação do mapa foi feita uma adaptação do exemplo proposto por Grinstead (2017). O código HTML gerado para visualização da grade foi ajustado para exibição em um *smarphone*. Foram feitas modificações no estilo CSS, para dar a impressão que as células estivessem homogeneamente concatenadas. O mapa exibido no aplicativo não é uma imagem, mas sim uma representação HTML de uma grade. A Figura 24 exemplifica o resultado obtido após ajuste do CSS.

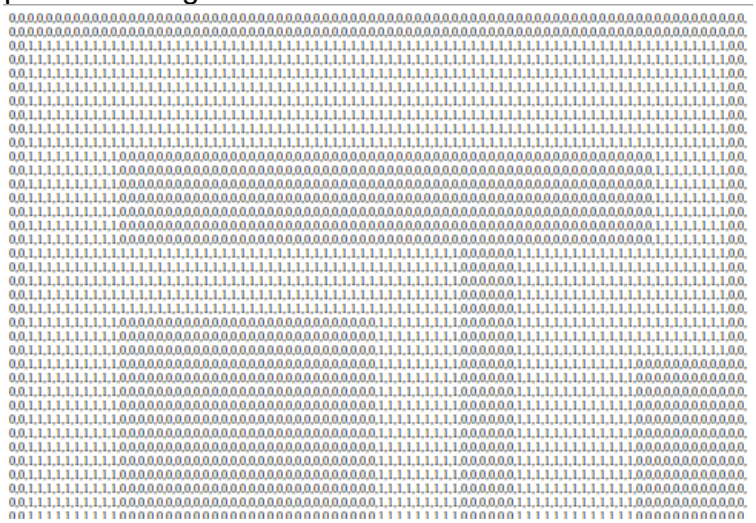
Figura 24 - Exemplo de uma grade e o mapa após ajuste do CSS



Fonte: Do autor (2017)

Para facilitar o desenho do mapa, criou-se um *script* JavaScript que lê uma imagem preto e branco e gera uma representação textual da mesma. Cada *pixel* da imagem é analisado, se for uma cor preta gera uma saída 0, caso for cor branca gera saída 1. Com isso é possível desenhar o ambiente simulado mais facilmente. A Figura 25 mostra o resultado gerado após a conversão de uma imagem. Esta representação textual é então carregada pela biblioteca javascript-astar, no formato de um vetor, que a partir disso cria o mapa 2D da aplicação.

Figura 25 - Resultado do *script* que gera o mapa a partir de imagem



Fonte: Do autor (2017)

O cálculo do caminho e a exibição no mapa é feito inteiramente pela biblioteca javascript-astar. A Figura 26 mostra um exemplo básico, ilustrando o seu uso. Após a definição do *array* (linha 1) representando o mapa, com base nas coordenadas início (linha 2) e fim (linha 3), o algoritmo faz a pesquisa e retorna o *array* contendo o melhor caminho entre os pontos.

Figura 26 - Código exemplo biblioteca javascript-astar

```
1.  var graph = new Graph([
        [1,1,1,1],
        [0,1,1,0],
        [0,0,1,1]
    ]);
2.  var start = graph.grid[0][0];
3.  var end = graph.grid[1][2];
4.  var result = astar.search(graph, start, end);
```

Fonte: Do autor (2017)

5.4 Portando OpenCV para *web*

O processo para permitir o uso da biblioteca OpenCV na *web*, utilizando WebAssembly, foi baseado em uma publicação do *blog* VideoCortex (SHAVIT, 2017).

No texto o autor descreve os passos e as dificuldades encontradas durante sua experiência, na criação de um experimento prático utilizando OpenCV. Inicialmente o artigo mostra como deve ser feita a compilação do OpenCV utilizando CMake, Ninja e o Emscripten como *toolchain.* Posteriormente é detalhada a construção de um projeto para processamento de arquivo de vídeo, utilizando o OpenCV, diretamente no navegador, portando o algoritmo C++ para web.

Foram realizadas algumas adaptações para aplicação neste trabalho. Basicamente houveram alterações para que fosse feito o processamento do vídeo proveniente da câmera, e para que o resultado da compilação gerasse o ".wasm". O APÊNDICE A descreve as configurações na preparação da biblioteca OpenCV, enquanto que o APÊNDICE B descreve o desenvolvimento do projeto C++ que é compilado para ".wasm".

5.5 Estimativa de movimento

Para esta fase de desenvolvimento, que envolve o WebAssembly, foram criados dois projetos. Um compreendendo a parte *web,* contendo os arquivos HTML e *scripts* JavaScript, responsável por obter fluxo de vídeo da câmera, carregar e compilar o módulo WebAssembly e exibir os resultados. O outro é o projeto C++ que gera o ".wasm", onde todo algoritmo de processamento dos *frames* e cálculo do movimento acontece.

O processo tem início na parte *web*, onde o fluxo de vídeo proveniente da câmera é adquirido. Foi desenvolvido um código JavaScript que aloca e copia o *array* que contém as informações da imagem, para dentro do espaço de memória do WebAssembly, retornando um ponteiro. Atualmente não é possível enviar e retornar estruturas de dados ou tipos complexos, como um *array* ou *string* diretamente de uma função WebAssembly. Somente são aceitos tipos primitivos como inteiros e números de ponto flutuante. Desta forma, este ponteiro é a referência para a imagem, que é passada como parâmetro da função chamada via WebAssembly, e que pode ser manipulado dentro da função C++.

Na Figura 27 é apresentado o código JavaScript onde é feita a chamada da função WebAssembly. A função “_vo_homography” é invocada na linha 368, recebendo como parâmetros: a largura da imagem, altura da imagem e ponteiro para matriz que representa a imagem. Além disso são enviados o número do *frame* atual e o tamanho do vetor que conterá o resultado da homografia. Entre as linhas 378 e 381 é feita a população de um vetor, cujos índices são obtidos a partir do ponteiro retornado pelo método “vo_homography”.

Figura 27 - Chamada função WebAssembly no JavaScript

```
363     let homography_matrix;
364     let homography_matrix_size = 9; // matrix 3x3
365     try {
366
367         // chama a função WebAssembly criada para calcular a homografia
368         homography_matrix = Module._vo_homography(img_data.width,      // largura da imagem
369                                                   img_data.height,    // altura da imagem
370                                                   fp.frame_bytes.byteOffset, // ponteiro referenciando a imagem
371                                                   frameIndex, // número do frame
372                                                   homography_matrix_size); // tamanho do vetor de resultados
373
374     } catch (e) {
375         throw e
376     }
377     // popula o array com o resultado da homografia obtido a partir da função WebAssembly
378     var homography = [];
379     for (let v = 0; v < homography_matrix_size; v++) {
380         homography.push(Module.HEAPF64[homography_matrix / Float64Array.BYTES_PER_ELEMENT + v])
381     }
```

Fonte: Do autor (2017)

A função C++ analisa a homografia entre os *frames*. Para isso são empregadas a funções da biblioteca OpenCV, que fazem a extração de pontos característicos da imagem, rastreamento de pontos e extração da matriz de homografia.

A Figura 28 mostra a função “vo_homography” no lado C++. Ela é invocada na parte *web*, na Figura 27 linha 368, e recebe os parâmetros enviados pelo JavaScript. Na função “vo_homography”, Figura 28, linha 184, é criada uma representação do *frame.* O tipo “Mat” é uma estrutura de dados do OpenCV que serve como um contêiner para as informações da imagem. Na linha 188 é feito um pré-processamento da imagem, uma conversão para escala de cinza, para facilitar o trabalho do algoritmo FAST. O método “computeHomography”, linha 191, foi criado para computar a homografia, ela recebe a imagem e o contador do número de *frames*, retornando a matriz de homografia. Esta matriz é utilizada para preencher o

vetor de retorno, entre as linas 194 e 203. Por fim, na linha 206 é criado o ponteiro para o primeiro elemento do vetor, utilizado como valor de retorno da função.

Figura 28 - Função vo_homography

```
177   double* EMSCRIPTEN_KEEPALIVE vo_homography(  int width,
178                                                int height,
179                                                cv::Vec4b* frame4b_ptr,
180                                                int frameIndex,
181                                                int matrix_size
182                                             ) try {
183         // cria uma matriz representado o frame, basada no ponteiro enviado
184         Mat frame(height, width, CV_8UC4, frame4b_ptr);
185         // matriz que recebe a imagem convertida para escala de cinza
186         Mat gray(height, width, CV_8UC3, Scalar(0,0,0));
187         // converte a imagem em escala de cinza
188         cv::cvtColor(frame, gray, CV_RGBA2GRAY);
189         // chama a função responsável por computr a homografia
190         cv::Mat homography;
191         homography = computeHomography(gray, frameIndex);
192
193         // cria um vetor que reacebe os valores da matriz de homografia
194         double values[matrix_size];
195         values[0] = homography.at<double>(0, 0);
196         values[1] = homography.at<double>(0, 1);
197         values[2] = homography.at<double>(0, 2);
198         values[3] = homography.at<double>(1, 0);
199         values[4] = homography.at<double>(1, 1);
200         values[5] = homography.at<double>(1, 2);
201         values[6] = homography.at<double>(2, 0);
202         values[7] = homography.at<double>(2, 1);
203         values[8] = homography.at<double>(2, 2);
204
205         // retorna o ponteiro para o vetor contendo os valores da homografia
206         auto arrayPtr = &values[0];
207         return arrayPtr;
208
209     } catch (std::exception const& e) {
```

Fonte: Do autor (2017)

Dentro da função “computeHomography”, Figura 29, a cada quinze *frames* provenientes da câmera, um é definido como base. Do *frame* base são extraídos e descritos pontos, através do algorimo FAST e ORB respectivamente. Estes pontos são rastreados nos *frames* subseguistes. Na Figura 30, os pontos com mais precisão na associação são utilizados para o cálculo da homografia através da função “findHomography” do OpenCV.

Figura 29 - Função computeHomograhy

```
47    cv::Mat computeHomography(cv::Mat& Image, int frameIndex) {
48
49        VO::RobustMatcher robustMatcher;
50        // limiar de intensidade do pixel , para ser considerado um canto (ponto caracteristico)
51        int fast_threshold = 20;
52        // selecionar os melhores x pontos
53        int points_retain = 150; //
54        bool nonmaxSuppression = true;
55
56        // seleciona um frame base a cada 15
57        if(frameIndex % 15 == 0) {
58
59            firstImage = Image;
60
61            // extrai os pontos com o algorimo FAST
62            cv::FAST(firstImage, FASTKeypoints1, fast_threshold, nonmaxSuppression);
63
64            // seleciona os melhores pontos encontrados
65            cv::KeyPointsFilter::retainBest(FASTKeypoints1, points_retain);
66
67            // esta função utiliza o ORB para descrever os pontos
68            robustMatcher.computeDescriptors(firstImage,FASTKeypoints1,orbDescriptors1);
69
70            if ( orbDescriptors1.empty() )
71            cvError(0,"MatchFinder","1st descriptor empty",__FILE__,__LINE__);
72
73            if(orbDescriptors1.type()!=CV_32F) {
74                orbDescriptors1.convertTo(orbDescriptors1, CV_32F);
75                orbDescriptors1.convertTo(orbDescriptors1_8U, CV_8U);
76            }
77
78        } else {
79
80            // se não é o frame base
81            secondImage = Image;
82        }
```

Fonte: Do autor (2017)

Figura 30 - Final da função computeHomography

```
146         std::vector<cv::Point2f> img1Keypoints;
147         std::vector<cv::Point2f> img2Keypoints;
148
149         for( int i = 0; i < good_matches.size(); i++ ) {
150             // extrai os pontos a partir das melhores associações
151             img1Keypoints.push_back( FASTKeypoints1[ good_matches[i].queryIdx ].pt );
152             img2Keypoints.push_back( FASTKeypoints2[ good_matches[i].trainIdx ].pt );
153         }
154
155         cv::Mat H = cv::findHomography(img1Keypoints,img2Keypoints,CV_RANSAC);
156
157         if(!H.empty()){
158             //std::cout<<"Homography Computed"<<std::endl;
159         } else {
160             std::cout<<" NO Homography "<<std::endl;
161         }
162
163         return H;
164
165     }
```

Fonte: Do autor (2017)

6 DISCUSSÃO DOS RESULTADOS

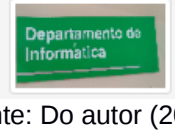
Neste capítulo são apresentados alguns dos resultados obtidos após ensaios com o reconhecimento baseado em imagem e odometria visual. A primeira simulação apresenta testes com a aplicação CBIRest. Algumas imagens de placas informativas, presentes em um prédio, foram retratadas em diferentes ângulos. Foram feitas pesquisas com imagens da mesma placa, só que em posições diferentes. O teste mostrou que o CBIRest conseguiu estabelecer uma similaridade entre as imagens, mesmo de pontos de vista distintos. A Figura 31 mostra o resultado de uma pesquisa por imagem. A placa identificada como “Suporte Informática”, no topo ao centro, foi utilizada como entrada. A lista de resultados mostra que imagem mais similar, com pontuação de 0.000211, é justamente a placa “Suporte Informática”, vista de um ângulo frontal. Do mesmo modo, a Figura 32 mostra o resultado para a placa “Segurança do trabalho”.

Figura 31 - Ensaio CBIRest, placa "Suporte Informática"

2		0.0002113485783820253	{"id":"2","storage":"Teste CBIRest"}
6		0.0001236547688145311	{"id":"6","storage":"Teste CBIRest"}
4		0.00009617792142498026	{"id":"4","storage":"Teste CBIRest"}
8		0.0000832187313784633	{"id":"8","storage":"Teste CBIRest"}

Fonte: Do autor (2017)

Figura 32 - Ensaio CBIRest, placa "Segurança do Trabalho"

8		0.000285664711920316	{"id":"8","storage":"Teste CBIRest"}
2		0.00026416797872481646	{"id":"2","storage":"Teste CBIRest"}
6		0.00011816730935486746	{"id":"6","storage":"Teste CBIRest"}
4		0.000016000000000000003	{"id":"4","storage":"Teste CBIRest"}

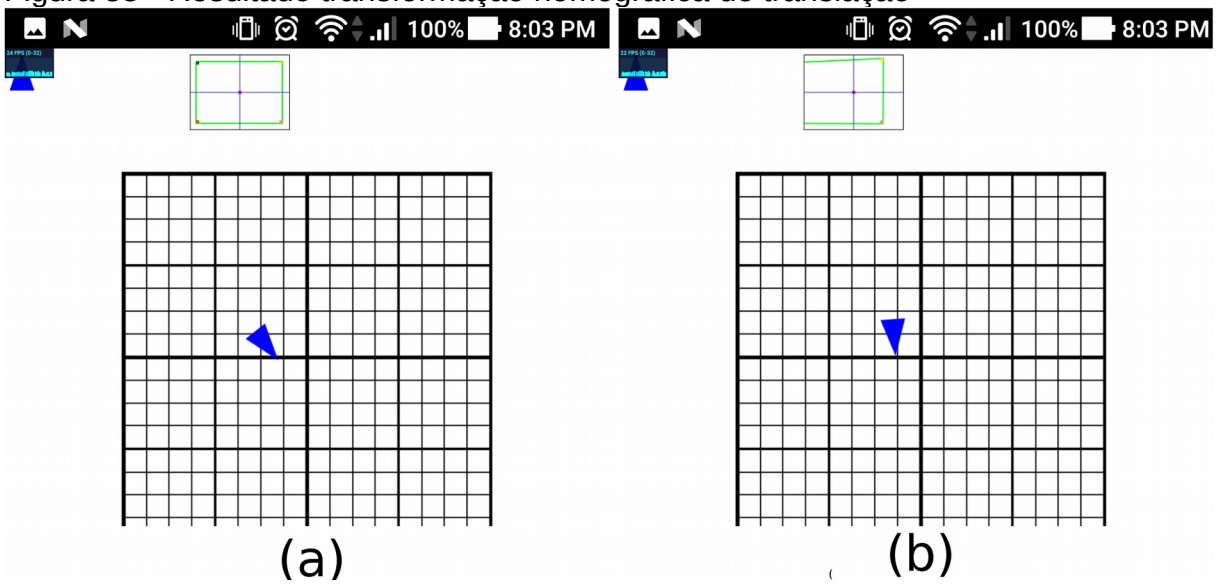
Fonte: Do autor (2017)

Em relação a odometria visual, foram avaliadas duas técnicas, utilizando funções da biblioteca OpenCV. Os testes foram realizados em laboratório, em um ambiente controlado, com iluminação e texturas adequadas, permitindo a extração e correspondência de 100 pontos em média, para cada quadro do vídeo.

A primeira tentativa baseou-se na homografia. Através das propriedades de transformação homográfica, de escala e translação, com o objetivo de estimar o movimento da câmera. No experimento, o vídeo da câmera foi adquirido com uma resolução de 160 x 120 *pixel.* O processamento atingiu em média 20 FPS, o que mostra a superioridade do WebAssembly, em relação a biblioteca JSFeat.

A transformação de translação permitiu verificar o deslocamento para direita ou esquerda. A Figura 33 mostra duas capturas de tela do *smartphone* exibindo a aplicação. Em um primeiro instante (a), a câmera está parada. No instante (b) o *smartphone* é rotacionado para direita. Neste momento o quadro de transformação homográfica, no topo, mostra o deslocamento estimado. Pode-se perceber que ele foi translado para o lado esquerdo, contrário ao movimento. No quadrado quadriculado ao centro é exibido um triângulo azul, mostrando o movimento.

Figura 33 - Resultado transformação homográfica de translação



Fonte: Do autor (2017)

O movimento de baseado na transformação de escala, para indicar movimento pra frente ou pra trás, não pode ser calculado com precisão, até o momento da entrega do trabalho. Alternativas, propostas como trabalho futuro, são

análise mais aprofundada sobre as transformações de escala, juntamente à decomposição da matriz homográfica em vetores de rotação e translação.

O segundo teste, com objetivo encontrar uma solução mais precisa, fundamentou-se no trabalho de Maidana (2015), com cerne no cálculo da matriz essencial. Foi adaptado o algoritmo para processamento em tempo real, analisando os *frames* conseguintes em uma espécie de janela deslizante de duas posições. Os testes realizados não puderam ser concluídos em tempo hábil até a entrega do trabalho.

Cabe também ressaltar o sucesso na portabilização do OpenCV para *web*. Com a utilização do Emscripten foi possível utilizar todas as funções da biblioteca, necessárias a implementação deste projeto. Isso indica que em breve, com a popularização do WebAssembly, outras bibliotecas C++ estarão disponíveis para criação de aplicações cada vez mais sofisticadas. Os ensaios realizados mostraram a sensibilidade do algoritmo de odometria em relação a qualidade das imagens captadas. Durante o deslocamento em áreas com pouca diferenciação de cor ou contraste, como paredes lisas, o cálculo da homografia não pode ser computada. Todo o processo de extração e comparação de pontos, quadro a quadro, requer um alto desempenho, desta maneira a execução em dispositivos mais antigos, com pouco poder de processamento, apresentaria um tempo de resposta insatisfatório ou inviável.

7 CONSIDERAÇÕES FINAIS

Com o objetivo de conceber uma alternativa para a navegação *indoor, o* presente trabalho apresentou uma proposta prática, para o uso da odometria visual e de sistemas de busca baseado em imagem. Paralelamente, a construção da solução, baseada no desenvolvimento *mobile* híbrido, permitiu pesquisar e explorar as características do WebAssembly. Além disso foi possível verificar o desempenho deste novo formato, no processamento de imagens relacionada a odometria visual, utilizando a biblioteca OpenCV , portando código C++ para *web*.

Durante a pesquisa foram apresentadas as dificuldades relacionadas a navegação *indoor*. Juntamente às tecnologias que vêm sendo desenvolvidas, para permitir o correto posicionamento de um agente, dentro de prédios e construções. Este trabalho teve como princípio o desenvolvimento de um aplicativo, que pudesse usar somente a câmera do *smartphone* como sensor de movimento e inferência da posição inicial. Além disso, a aplicação apresentou uma forma de *wayfinding,* com a exibição do melhor trajeto entre dois pontos, em um mapa.

Foi possível verificar a acuracidade da aplicação CBIRest para pesquisa baseada em imagem. Empregada no reconhecimento do ponto inicial de navegação, a CBIRest demonstrou acerto nos testes de reconhecimento, utilizando marcos e placas presentes no ambiente. O uso da odometria visual em um *smartphone*, conforme ensaios realizados, mostrou-se inviável em um primeiro momento. As técnicas visuais, para estimativa do movimento, são muito suscetíveis ao ambiente e necessitam um poder de processamento considerável. Depende-se muito da

iluminação e das texturas presentes no espaço, para que o algoritmo de extração e rastreio de pontos possa ter efetividade.

Simultaneamente, o cálculo para estimar o movimento, a partir da homografia, mostrou-se inconclusivo. São necessários estudos mais aprofundados sobre o assunto. Algumas funções do OpenCV, relacionadas a transformações projetivas poderiam ser usadas, na tentativa de melhorar os resultados. Outro ponto negativo é dificuldade de estimar a escala de movimento, empregando apenas uma câmera.

Contudo, o experimento proporcionou um estudo e implementação do WebAssembly em um caso prático. Esta nova tecnologia, que está sendo desenvolvida em conjunto, pelas grandes empresas desenvolvedoras de *browsers,* promete elevar a web a um outro nível. Foi possível constatar, no ato, o aumento de desempenho no processamento da odometria visual baseada na homografia, após a implementação com o WebAssembly. Outro ponto interessante foi a possibilidade de trazer para web, todas as funcionalidades da biblioteca OpenCV, escrita em C++.

Propõe-se, como trabalhos futuros, o aperfeiçoamento do processo, empregando a odometria visual inercial. Esta técnica consiste na fusão de dados vindos dos sensores giroscópio e acelerômetro juntamente para auxiliar no processamento visual. A odometria visual-inercial fornece maior precisão, principalmente para estimar a escala e complementar as medidas de deslocamento, nos casos onde a odometria visual, por si só, não consegue resolver. Sugere-se também a revisão do algoritmo apresentado por Maidana (2015) e aperfeiçoamento dos resultados obtidos com a análise homográfica de movimento. Em relação ao desempenho, a odometria visual poderia ser testada através de uma aplicação Android nativa, utilizando o OpenCV SDK.

REFERENCIAS BIBLIOGRÁFICAS

AKHMADEEV, Foat. **Computer Vision for the Web:** Unleash the power of Computer Vision algorithms in JavaScript to develop vision-enabled web content. 1. ed.Birmingham: Packt Publishing Ltd., 2015. 116 p.

APACHE. **Cordova**. Disponível em: <https://cordova.apache.org/>. Acesso em: 03 mai. 2017.

AQEL, Mohammad O. A. et al. **Review of visual odometry:** types, approaches, challenges, and applications. [S.l.: s.n.], 2016. 2 p. v. 7.Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5084145/>. Acesso em: 24 abr. 2017.

ARTHUR, Paul; PASSINI, Romedi. **Wayfinding:** People, Signs, and Architecture. 1. ed. New York, EUA: McGraw-Hill Book Co, 1992. 238 p.

ASHWINI, B. M.; USHA, J. . **Location Based Services:** Positioning Techniques and its Applications. International Journal of Application or Innovation in Engineering & Management, Bangalore, v. 1, n. 1, jan. 2014. Disponível em: <http://www.ijaiem.org/volume3issue1/IJAIEM-2014-01-20-042.pdf>. Acesso em: 11 abr. 2014.

BHARTI, Santosh; WADHWA, Lalit. **Content Based Image Retrieval Components:** Color, Shape and Texture . Dezembro 2013. 303 p. v. 1.Disponível em: <http://ijaegt.com/wp-content/uploads/2013/12/IJAEGT-309156-SNT-303-308.pdf>. Acesso em: 03 maio 2017.

BINARYEN. **Compiler infrastructure and toolchain library for WebAssembly, in C++**. Disponível em: <https://github.com/WebAssembly/binaryen>. Acesso em: 02 nov. 2017.

BMVA. **What is computer vision?**. 2017. Disponível em:<http://www.bmva.org/visionoverview>. Acesso em: 16 abr. 2017.

BUCZKOWSKI, Aleksander. **Location-Based Services** . Geo Awesomeness, 2012. Disponível em: <http://geoawesomeness.com/knowledge-base/location-based-services/location-based-services-components/>. Acesso em: 11 abr. 2017.

CAN I use WebAssembly?. Disponível em: <http://caniuse.com/#feat=wasm>. Acesso em: 01 nov. 2017.

CBIREST. **Presentation**. Disponível em <http://loic911.github.io/cbirest/>. Acesso em: 12 mai. 2017.

CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a estruturas de dados:** Com técnicas de programação em C. 2. ed. Rio de Janeiro: Elsevier, 2016. 408 p.

CLARK, Lin. **Where is WebAssembly now and what's next?**. Disponível em: <https://hacks.mozilla.org/2017/02/where-is-webassembly-now-and-whats-next/>. Acesso em: 30 out. 2017a.

CLARK, Lin. **Creating and working with WebAssembly module**. Disponível em: <https://hacks.mozilla.org/2017/02/creating-and-working-with-webassembly-modules/>. Acesso em: 30 out. 2017b.

CMAKE About . Disponível em: <https://cmake.org/overview/>. Acesso em: 02 nov. 2017.

DALLING, Tom. **Explaining Homogeneous Coordinates & Projective Geometry**. Disponível em: <https://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry/>. Acesso em: 01 nov. 2017.

DAVIES, E. R. **Computer and Machine Vision:** Theory, Algorithms, Practicalities. 4. ed. Waltham, EUA: Academic Press, 2012. 912 p.

DOS SANTOS, M. C. **Revisão de Conceitos em Projeção, Homografia, Calibração de Câmera, Geometria Epipolar, Mapas de Profundidade e Varredura de Planos**. 2012. Disponível em: <http://www.ic.unicamp.br/~rocha/teaching/2012s1/mc949/aulas/additional-material-revision-of-concepts-homography-and-related-topics.pdf>. Acesso em: 23 nov. 2017.

EMSCRIPTEN. Disponível em: <http://kripken.github.io/emscripten-site/index.html>. Acesso em: 02 nov. 2017.

FACEBOOK desafia Google com novo serviço de publicidade. **Valor**, New York, EUA, 18 fev. 2015. Disponível em: <http://www.valor.com.br/empresas/3913962/facebook-desafia-google-com-novo-servico-de-publicidade>. Acesso em: 07 abr. 2017.

FOLTZ, Mark A. **Designing Navigable Information Spaces**. 1998. 130 p. Dissertação (Mestrado)- Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, St. Louis, 1998. Disponível

em:<http://rationale.csail.mit.edu/publications/Foltz1998Designing.pdf>. Acesso em: 08 maio 2017.

GERSTWEILER, Georg; VONACH, Emanuel; KAUFMANN, Hannes. **HyMoTrack:** A Mobile AR Navigation System for Complex Indoor Environments. Viena, Aústria: [s.n.], 2015. 19 p. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4732050/>. Acesso em: 12 maio 2017.

GIL, Antônio Carlos. **Métodos e Técnicas de Pesquisa Social**. 6. ed. São Paulo, SP: Atlas, 2008. 216 p.

GITHUB. **Info**. 2017. Disponível em: <http://githut.info/> Acesso em: 03 mai. 2017a.

GITHUB. **rockchico/AppTcc**. 2017. Disponível em: <https://github.com/rockchico/AppTcc> Acesso em: 13 dez. 2017b.

GITHUB. **rockchico/AppTcc-wasm**. 2017. Disponível em: <https://github.com/rockchico/AppTcc-wasm> Acesso em: 13 dez. 2017c.

GOOGLE. **Maps**. 2017. Disponível em: <https://www.google.com.br/maps> Acesso em: 02 mai. 2017.

GRINSTEAD, Brian. **A* Search Algorithm in JavaScript (Updated)**. Disponível em: <https://briangrinstead.com/blog/astar-search-algorithm-in-javascript-updated/>. Acesso em: 31 out. 2017.

HASSABALLAH, M. ; ABDELMGEID, Aly Amin; ALSHAZLY, Hammam A. **Image Features Detection, Description and Matching**. In: AWAD, Ali Ismail; HASSABALLAH, Mahmoud. Image Feature Detectors and Descriptors: Foundations and Applications. 1. ed. [S.l.]: Springer, 2016. cap. 1, p. 11-45.

HARTLEY, R. ; ZISSERMANN, A. **Multiple View Geometry in Computer Vision**, 2 ed. Cambridge University Press, 2004.

HU, Feng. **Emerging Techniques in Vision-based Indoor Localization**. 2015. Monografia – University of New York. Disponível em: <https://pdfs.semanticscholar.org/050b/e5a5f98137270375275f7993ad038ad3060b.pdf>. Acesso em: 13 abr. 2017

HUANG, Yewei et al. **Image-Based localization for indoor enviroments using mobile phone**. Toquio, Japão: [s.n.], 2015. 5 p.Disponível em: <http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-4-W5/211/2015/isprsarchives-XL-4-W5-211-2015.pdf>. Acesso em: 12 maio 2017.

INSPIRIT. **JSFeat**. 2017. Disponível em: <https://inspirit.github.io/jsfeat/> Acesso em: 03 mai. 2017.

KANARIS, Loizos et al. **Fusing Bluetooth Beacon Data with Wi-Fi Radiomaps for Improved Indoor Localization**. *Sensors.* 2017. Disponível em: <http://www.mdpi.com/1424-8220/17/4/812>. Acesso em: 13 abr. 2017.

KARIMI, Hassan. **Indoor Wayfinding and Navigation**. Boca Raton, EUA: CRC Peess, 2015. 266 p.

KRIG, Scott. Computer Vision Metrics: Survey, Taxonomy, and Analysis. 1. ed. Apress, 2014. 508 p.

LIU, Jingbin et al. **A Hybrid Smartphone Indoor Positioning Solution for Mobile LBS**. Sensors. 2012. Disponível em: <http://www.mdpi.com/1424-8220/12/12/17208>. Acesso em: 13 abr. 2014.

LLVM The Compiler Infrastructure. Disponível em: <https://llvm.org/>. Acesso em: 02 nov. 2017.

LOPES, Sérgio. Aplicações mobile híbridas com Cordova e PhoneGap. 1. ed. São Paulo, SP: Casa do Código, 2016. 192 p.

MAIDANA, Renan G. **Odometria Visual para Robótica Móvel**. 2015. 36 p. Monografia (Bacharel)- Eng. Controle e automação, PUC-RS, Porto Alegre, 2015. 1.

MAUTZ, Rainer. **Indoor Positioning Technologies**. 2012. Monografia – Institute of Geodesy and Photogrammetry, Department of Civil, Environmental and Geomatic Engineering, Zurich Federal Institute of Technology , fev.2012. Disponível em: <http://e-collection.library.ethz.ch/eserv/eth:5659/eth-5659-01.pdf>. Acesso em: 11 abr. 2017

MOZILLA. **JavaScript**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 07 abr. 2017a.

MOZILLA. **WebAssembly Concepts**. Disponível em: <https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts>. Acesso em: 30 out. 2017b.

MOZILLA. **WebAssembly format**. Disponível em: <https://research.mozilla.org/webassembly/>. Acesso em: 30 out. 2017c.

NINJA. Disponível em: <https://ninja-build.org/>. Acesso em: 02 nov. 2017.

NISTÉR, David. **An efficient solution to the five-point relative pose problem**. [S.l.]: IEEE, 2004. 12 p. Disponível em:<https://pdfs.semanticscholar.org/2621/597b539f2930df0c8aafe0464b36f6876067.pdf>. Acesso em: 15 abr. 2017.

NISTÉR, David; NARODITSKY, Oleg; BERGEN, James. **Visual Odometry**. Princeton, EUA: IEEE, 2004. 12 p. Disponível em:<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.5767&rep=rep1&type=pdf>. Acesso em: 15 abr. 2017.

NIXON, Mark S.; AGUADO, Alberto S. **Feature Extraction and Image Processing for Computer Vision**. Londres, Inglaterra: Academic Press, 2012. 632 p.

NÚMERO de smartphones em uso no Brasil chega a 168 milhões, diz estudo. **Folha**, São Paulo, 15 abr. 2016. Disponível em: <http://www1.folha.uol.com.br/mercado/2016/04/1761310-numero-de-smartphones-em-uso-no-brasil-chega-a-168-milhoes-diz-estudo.shtml>. Acesso em: 07 abr. 2017.

O'GRADY, J.V ; O´GRADY, K.V. **The Information Design Handbook**. Cincinnati, EUA: How Books, 2008.

OPENCV. **OpenCV**. 2017a. Disponível em: <http://opencv.org/>. Acesso em: 07 abr. 2017.

OPENCV. **About**. 2017b. Disponível em: <http://opencv.org/about.html>. Acesso em: 31 out. 2017.

OPENSTREETMAP. **OpenStreetMap**. 2017. Disponível em: <https://www.openstreetmap.org> Acesso em: 02 mai. 2017.

PHONEGAP. **Build amazing mobile apps powered by open web tech**. . 2017. Disponível em: <http://phonegap.com/> Acesso em: 18 mai. 2017.

REDMONK. **The RedMonk Programming Language Rankings:** January 2017. 17 mar. 2017. Disponível em: <http://redmonk.com/sogrady/2017/03/17/language-rankings-1-17/> Acesso em: 03 mai. 2017.

RYAN, Dan. **E - learning Modules:** DLR Associates Series. 1. ed. Bloomington: AuthorHouse, 2012. 501 p.

SALEH, Hazem et al. **Mobile Application Development:** JavaScript Frameworks. 1. ed. Birmingham, Reino Unido: Packt Publishing, 2016. 599 p

SANTHI, V. ; ACHARJYA, D. P. **Emerging Technologies in Intelligent Applications for Image and Video Processing**. 1. ed. Hersey, EUA: IGI Global,2016. 518 p. v. 1.

SATALICH, G.A., **Navigation and Wayfinding in Virtual Reality:** Finding Proper Tools and Cues to Enhance Navigation Awareness, Master’s Thesis, University of Washington, 1995.

SCARAMUZZA, Davide; FRAUNDORFER, **Friedrich. Visual Odometry Part I:** The First 30 Years and Fundamentals. 4. ed. [S.l.]: IEEE Robotics And Automation Magazine, 2011. 80 p. v. 18. Disponível em: <http://rpg.ifi.uzh.ch/docs/VO_Part_I_Scaramuzza.pdf>. Acesso em: 26 abr. 2017.

SCARAMUZZA, Davide; FRAUNDORFER, Friedrich. **Visual Odometry Part II:** Matching, Robustness, Optimization, and Applications. 2. ed. [S.l.]: IEEE Robotics

And Automation Magazine, 2012. 80 p. v. 19. Disponível em: <http://rpg.ifi.uzh.ch/docs/VO_Part_II_Scaramuzza.pdf>. Acesso em: 26 abr. 2017.

SCHILLER, Jochen; VOISARD, Agnès. **Location-Based Services**. 1. ed. São Francisco, EUA: Morgan Kaufmann, 2004. 255 p.
SHAVIT, Adi. **OpenCV Web Apps:** OpenCV comes to the web!. Disponível em: <http://videocortex.io/2017/opencv-web-app/>. Acesso em: 03 jul. 2017.

SHELLEY, Michael Andrew. **Monocular Visual Inertial Odometry on a Mobile Device**. 2014. 92 p. Tese (Mestrado em Informática), Universidade de Munique, Munique, Alemanha; 2014. Disponível em: <https://vision.in.tum.de/_media/spezial/bib/shelley14msc.pdf>. Acesso em:26 abr. 2017.

SIEGWART, Roland; NOURBAKHSH, Illah; SCARAMUZZA, Davide. **Introduction to Autonomous Mobile Robots:** Intelligent Robotics and Autonomous Agents series. 2. ed. Londres, Reino Unido: MIT Press, 2011. 472 p.

SMITH, Peter. **Software Build Systems:** Principles and Experience. 1. ed. Boston, EUA: Pearson, 2011. 624 p.

SPRINGER, Thomas. **MapBiquitous:** An Approach for Integrated Indoor/Outdoor Location-Based Services. In: MobiCASE: International Conference on Mobile Computing, Applications, and Services, 3., 2011, Los Angeles, CA, USA. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering , vol 95. Belim: Springer, 2012. p. 80.

STACKOVERFLOW. **Faça parte da comunidade Stack Overflow**. 2017. Disponível em: <https://stackoverflow.com/> Acesso em: 18 mai. 2017.

STOJKSKA, Biljana R.; KOSOVIC, Ivana N.; JAGUST, Tomislav. **A Survey of Indoor Localization Techniques for Smartphones**. In: 8th International Conference ICT Innovations 2016, 2016, Macedonia, Proceedings... 12 p. Disponível em: <https://www.academia.edu/31288343/A_Survey_of_Indoor_Localization_Techniques_for_Smartphones>. Acesso em: 11 abril 2017.

STERLING, Greg. **Magnectic Positioning:** The arrival of ‘indoor GPS’. jun 2014. Disponível em: <http://www.indooratlas.com/wp-content/uploads/2016/03/magnetic_positioning_opus_jun2014.pdf>. Acesso em: 11 abr. 2017.

SUDHAKARAN, **Supreeth:** Streetfight Moves INDOORS To Map the Last Three Feet.
Geospatial World, Noida, India, v. 5, n. 1, p. 42, aug 2014. Disponível em: <https://www.geospatialworld.net/magazine/august-2014/>. Acesso em: 07 abr. 2017.

SZELISKI, Richard. **Computer Vision:** Algorithms and Applications. 1. ed. London: Springer-Verlag, 2011. 812 p.

WANG, Xi et al. **An Indoor Positioning Method for Smartphones Using Landmarks and PDR**. *Sensors* 2016. Disponível em: <http://www.mdpi.com/1424-8220/16/12/2135>. Acesso em: 13 abr. 2014.

WEBASSEMBLY. Disponível em: <https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts>. Acesso em: 30 out. 2017.

WERNER, Martin. **Indoor Location-Based Services:** Prerequisites and Foundations. Suiça: Springer, 2014. 233 p.

TORRES, Ricardo; FALCÃO, Alexandre.**Content-Based Image Retrieval:** Theory and Applications. 1. ed. [S.l.: s.n.], 2006. 166 p. v. 1. Disponível em: <http://www.ic.unicamp.br/~rtorres/mo805R_13s1/2006-Torres2006RITA.pdf>. A cesso em: 03 maio 2017.

TRACKINGJS. **tracking.js:** A modern approach for Computer Vision on the web.2017. Disponível em: <https://trackingjs.com/> Acesso em: 03 mai. 2017.

VENDA de notebooks supera a de desktops, diz Abinee. **EXAME**, São Paulo, 09 dez. 2010. Disponível em: <http://exame.abril.com.br/tecnologia/venda-de-notebooks-supera-a-de-desktops-diz-abinee/>. Acesso em: 07 abr. 2017.

VILLANUEVA-ESCUDERO, Carla et al. **Monocular visual odometry based Navigation for a differential mobile robot with Android OS**. Cidade do México: [s.n.], 2014. 12 p. Disponível em: <http://www.micai.org/2014/pre-print/papers/88560275.pdf>. Acesso em: 12 maio 2017.

VISUAL Odometry on the Mars Exploration Rovers. Pasadena, EUA: [s.n.], 2005. Disponível em:<https://trs.jpl.nasa.gov/bitstream/handle/2014/37431/05-0761.pdf?sequence=1&isAllowed=y>. Acesso em: 24 abr. 2017.

APÊNDICE A – Preparação biblioteca OpenCV

O primeiro passo diz respeito a construção das bibliotecas estáticas do OpenCV com o CMake, Ninja e Emscripten. Para isso foram realizados as seguintes ações:

1) Fazer o download e descompactar as bibliotecas OpenCV 3.3.0 e módulos extra OpenCV Contrib:

```
francisco@netuno ~/Documentos/TCC $ wget https://github.com/opencv/opencv/archive/3.3.0.zip
```

```
francisco@netuno ~/Documentos/TCC $ unzip 3.3.0.zip
```

```
francisco@netuno ~/Documentos/TCC $ wget https://github.com/opencv/opencv/archive/3.3.0.zip
```

```
francisco@netuno ~/Documentos/TCC $ unzip 3.3.0.zip
```

2) Instalação CMake:

```
francisco@netuno ~/Documentos/TCC $ apt-get install cmake
```

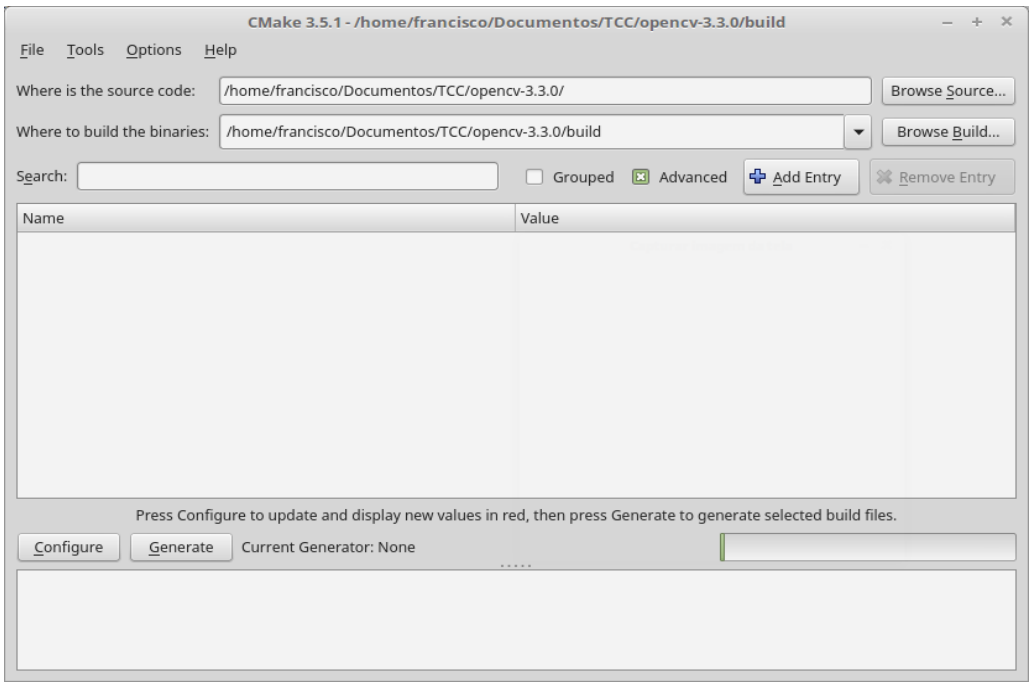
3) Instalação Ninja:

```
francisco@netuno ~/Documentos/TCC $ apt-get install ninja
```

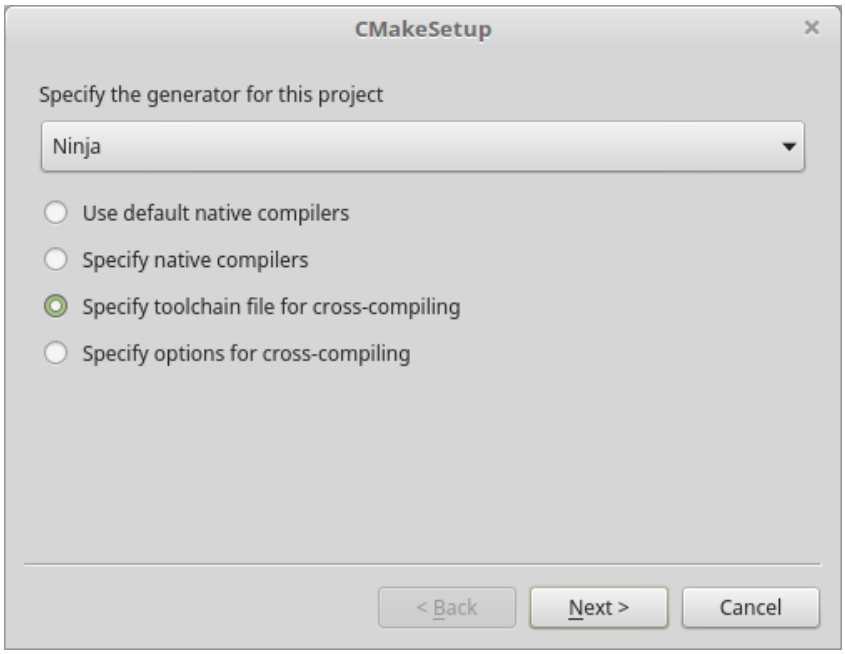
4) Instalação Emscripten: O Emscripten possui uma versão *portable, as instruções para instalação estão em: https://kripken.github.io/emscripten-site/docs/getting_started/downloads.html*

5) Configurar sistema de *build*: Abrir o CMake, configurar o projeto OpenCV utilizando o ninja e selecionando o *toolchain* Emscripten.

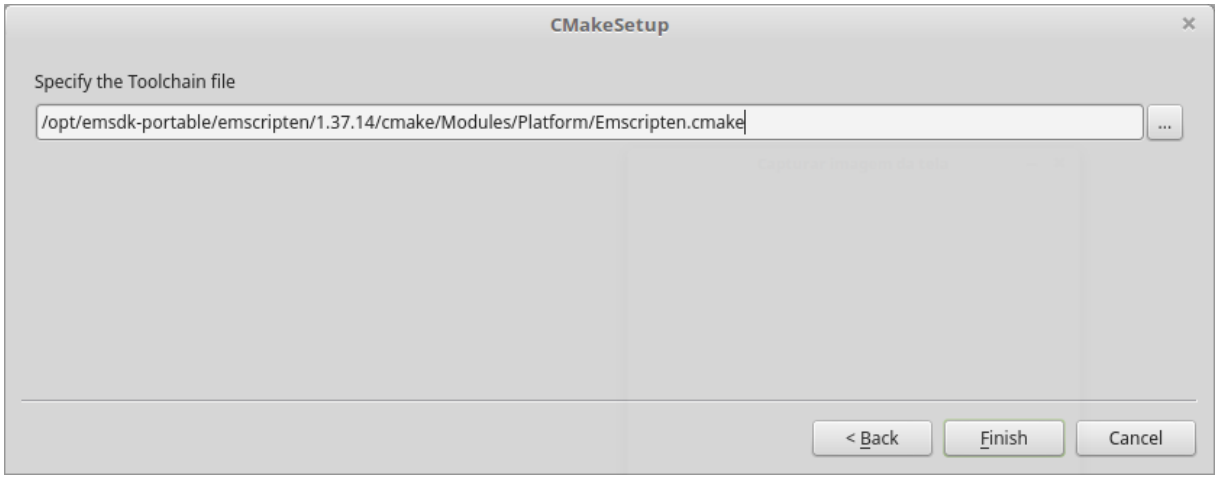
- Neste passo são necessários informar o local dos arquivos fonte da biblioteca OpenCV e o diretório onde serão gerados a saída compilada. Após isso deve ser feita a configuração do projeto, clicando no botão “Configure”.



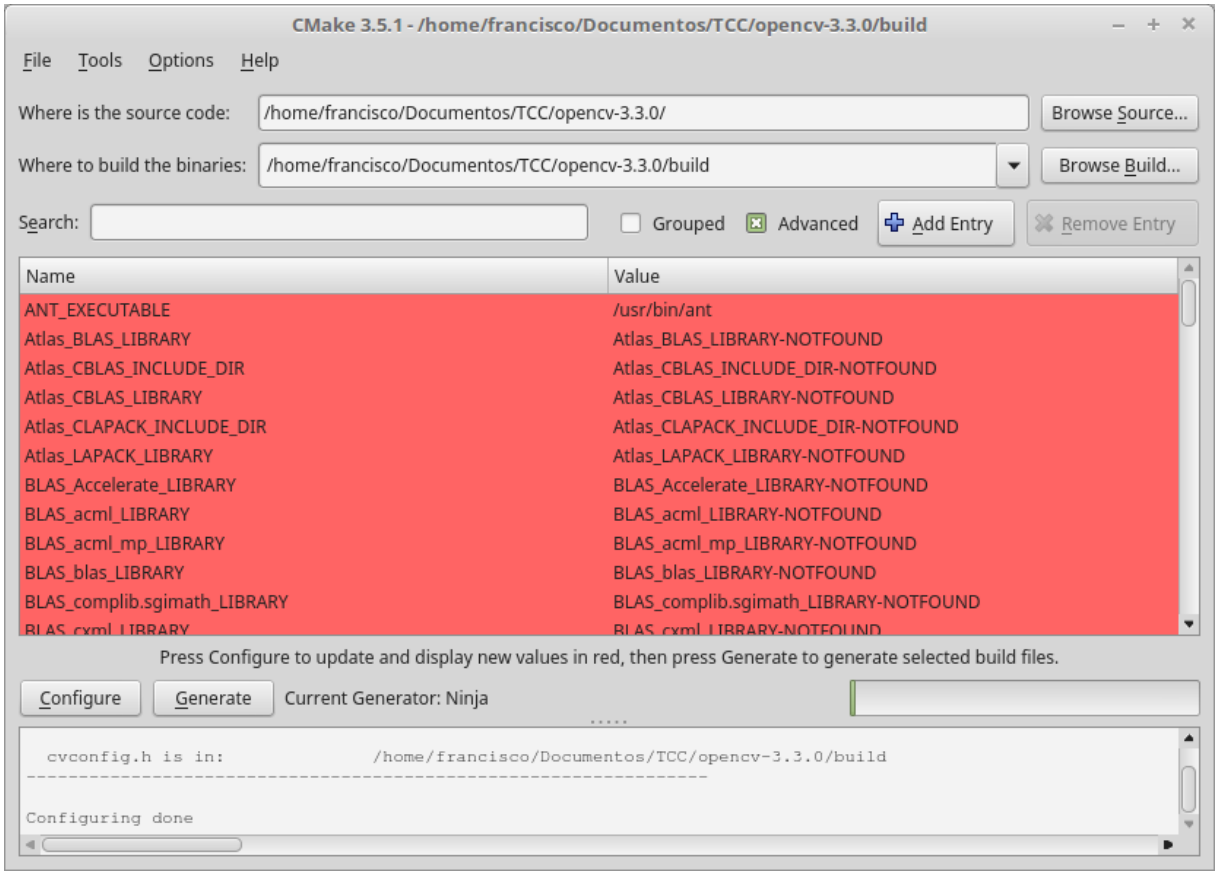
- Na tela de configuração, selecionar o o Ninja como gerador para este projeto. Marcar também a opção para especificar um *toolchain* de compilação.



- Especificar o caminho para o Emscripten *toolchain.* Ele é fornecido juntamente com a instalação do Emscripten Portable.



- Após isso o CMake fará alguns testes. Se não ocorrerem erros o projeto pode ser configurado.



- Neste experimento foram feitas as seguintes alterações nas diretivas de compilação:

- Foram habilitados:

BUILD_opencv_*

BUILD_JPEG

BUILD_PNG

- Desabilitados

BUILD_DOCS

BUILD_opencv_freetype

BUILD_opencv_biospired

BUILD_EXAMPLES

BUILD_FAT_JAVA_LIB

BUILD_IPP_IW

BUILD_PACKAGE

BUILD_PERF_TESTS

BUILD_SHARED_LIBS

BUILD_TESTS

BUILD_WITH_DEBUG_INFO

CV_ENABLE_INTRINSICS

WITH_PTHREADS_PF

- Para as diretivas CPU_BASELINE e CPU_DISPATCH devem ser selecionas as opções em branco.

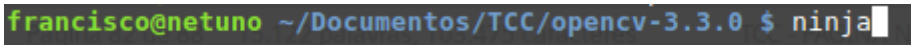
- Mudar a diretiva CMAKE_BUILD_TYPE para “Release”

- Configurar as opções do compilador Emscripten CMAKE_CXX_FLAGS and CMAKE_C_FLAGS para:

--llvm-lto 1 --bind -s WASM=1 -s ALLOW_MEMORY_GROWTH=1 -s DISABLE_EXCEPTION_CATCHING=0 -s ASSERTIONS=2 --memory-init-file 0 -O3

- Setar a variável OPENCV_EXTRA_MODULES_PATH com o diretório onde estão os módulos extras

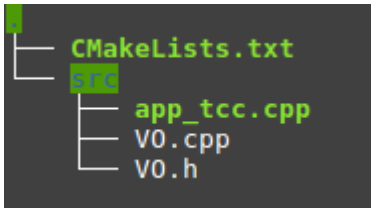
Após as alterações o projeto deve ser reconfigurado, clicando no botão “Configure” novamente. Quando o processo estiver pronto basta clicar no botão “Generate”. Para finalizar basta rodar o comando ninja dentro da pasta OPENCV_DIR/build. Com isso as bibliotecas OpenCV estarão compiladas para uso.



APÊNDICE B – Projeto C++ utilizando o OpenCV

Este passo a passo tem o objetivo de descrever o projeto C++ e a geração do “.wasm” que é carregado pelo navegador. Serão apresentadas o processo de configuração e compilação do projeto C++ , até a integração e uso do WebAssembly em uma página *web*. Da mesma forma que o processo de compilação apresentado no APÊNDICE A, este roteiro necessita a instalação do CMake, Ninja e Emscripten.

1) Criação do projeto, com a seguinte estrutura:

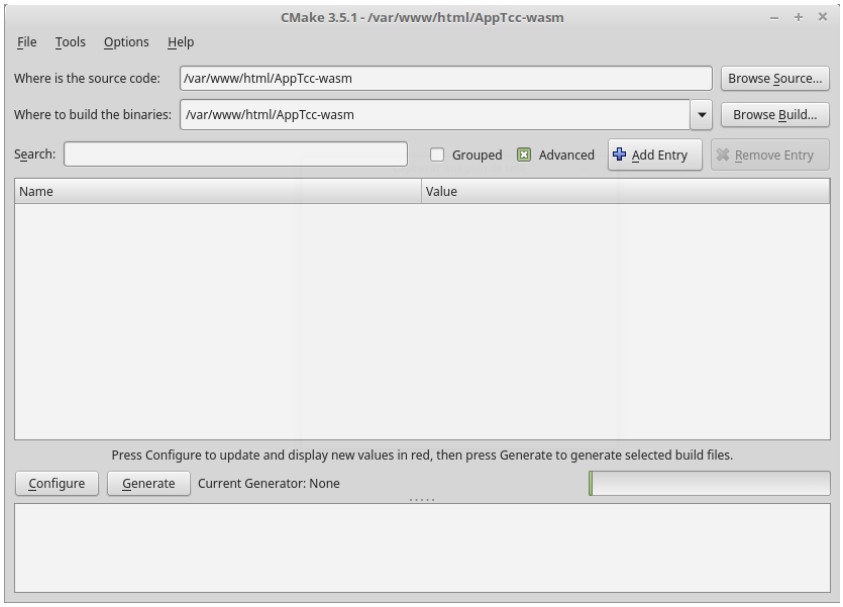


- **CmakeLists.txt:** contém as regras para construção do projeto
- **app_tcc.cpp:** arquivo principal que contém a função que será chamado via JavaScript, responsável por fazer o processamento dos *frames* do vídeo
- **VO.cpp:** contém a classe com funções relacionadas a analise e rastreio de pontos nas imagens
- **VO.h:** *headers* da classe contida no arquivo VO.cpp

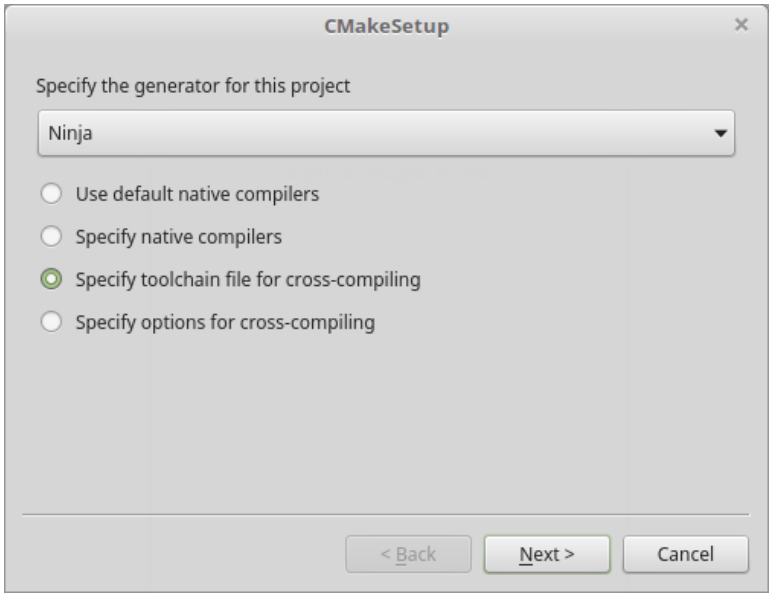
2) Configuração do sistema de *build:* O arquivo CmakeLists.txt contém as regras para geração do projeto utilizando o CMake, onde são definidas dependências, bibliotecas e o executável gerado.

```
project (AppTccWasm) # nome do projeto
cmake_minimum_required(VERSION 3.0)

set(OpenCV_STATIC ON)
find_package(OpenCV REQUIRED) # biblioteca OpenCV é requerida
include_directories(${OpenCV_INCLUDE_DIRS} .)

add_library(app_tcc_lib src/VO.cpp src/VO.h) # adiciona bibliotecas do projeto

if (EMSCRIPTEN)

    add_executable (app_tcc_wasm src/app_tcc.cpp) # arquivo executável

    target_link_libraries(app_tcc_wasm ${OpenCV_LIBS} app_tcc_lib) # inclui bibliotecas OpenCV no projeto

    # após a compilação, copia o arquivo app_tcc_wasm.js para a pasta do projeto web
    add_custom_command(TARGET app_tcc_wasm POST_BUILD
                       COMMAND ${CMAKE_COMMAND} -E copy_if_different
                       ${CMAKE_CURRENT_BINARY_DIR}/app_tcc_wasm.js
                       /var/www/html/AppTcc/www/testes/teste-wasm/app_tcc_wasm.js)

    # após a compilação, copia o arquivo app_tcc_wasm.js para a pasta do projeto web
    add_custom_command(TARGET app_tcc_wasm POST_BUILD
                       COMMAND ${CMAKE_COMMAND} -E copy_if_different
                       ${CMAKE_CURRENT_BINARY_DIR}/app_tcc_wasm.wasm
                       /var/www/html/AppTcc/www/testes/teste-wasm/app_tcc_wasm.wasm)

endif()

if(UNIX)

    # flags compilação Emscripten
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++1z --llvm-lto 1 --bind -s WASM=1 ")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -s ALLOW_MEMORY_GROWTH=1 -s DISABLE_EXCEPTION_CATCHING=0 ")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -s ASSERTIONS=2 ")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -s DISABLE_EXCEPTION_CATCHING=0 -s NO_FILESYSTEM=1 ")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -s NO_EXIT_RUNTIME=1 ")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} --memory-init-file 0 -O3")

    if(DEFINED CMAKE_BUILD_TYPE)
        SET(CMAKE_BUILD_TYPE ${CMAKE_BUILD_TYPE})
    else()
        SET(CMAKE_BUILD_TYPE Release)
    endif()

endif()
```

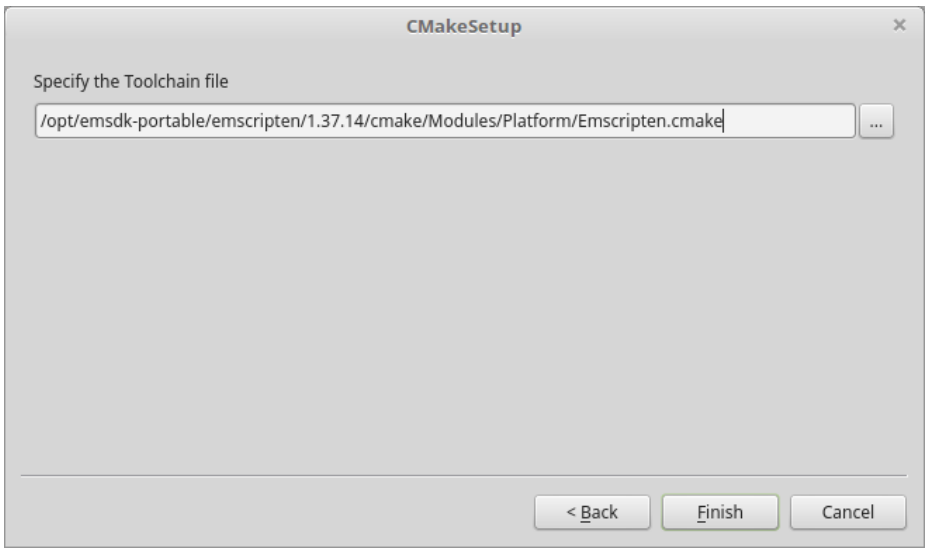
- A configuração e geração dos arquivos de configuração com CMake precisa ser feito apenas na inicialização do projeto. Inicialmente, com o CMake aberto, basta informar o diretório fonte e o diretório onde será construído o projeto. Neste exemplo foi definido o diretório “/var/www/html/AppTcc-wasm”.



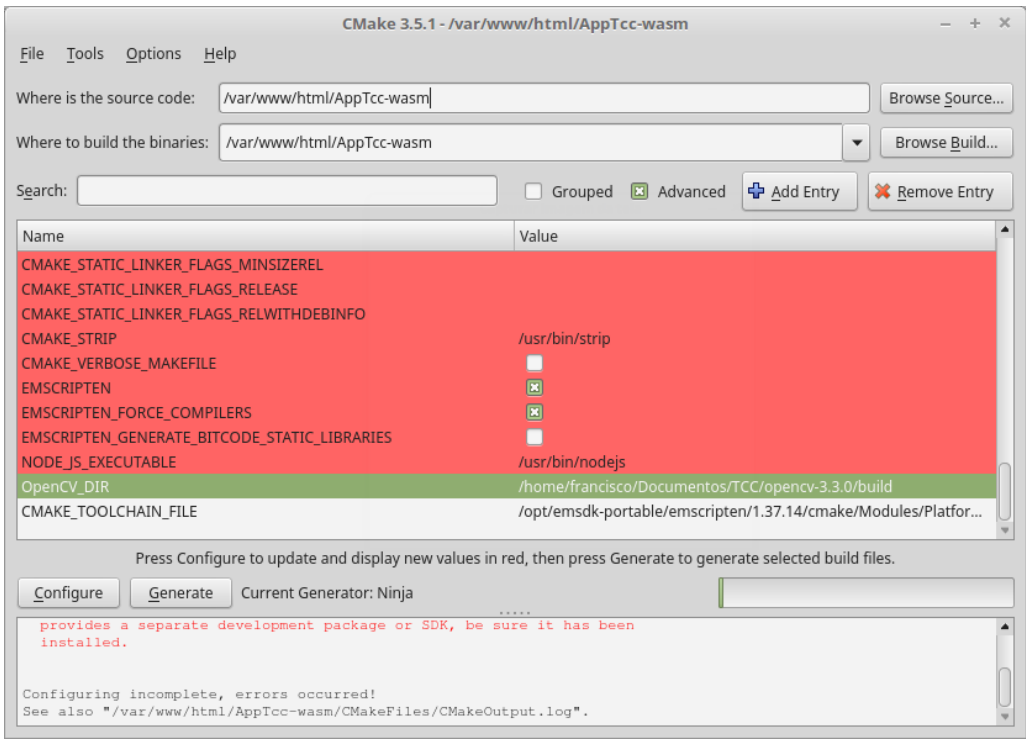
- Em seguida, ao clicar em “configure”, é preciso especificar o ”Ninja” como alvo para geração do projeto e marcar a opção que permite especificar um *toolchain* para compilação.



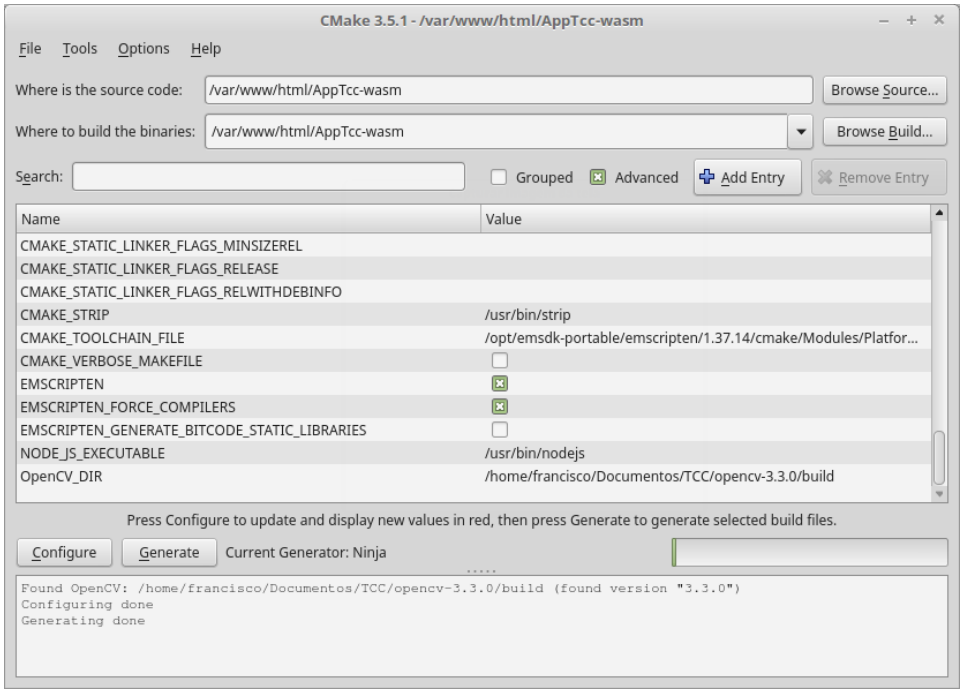
- Aqui é preciso especificar o Emscripten como *toolchain* para compilação, informando o caminho do arquivo “Emscripten.cmake”. Este arquivo encontra-se no diretório do Emscripten Portable.



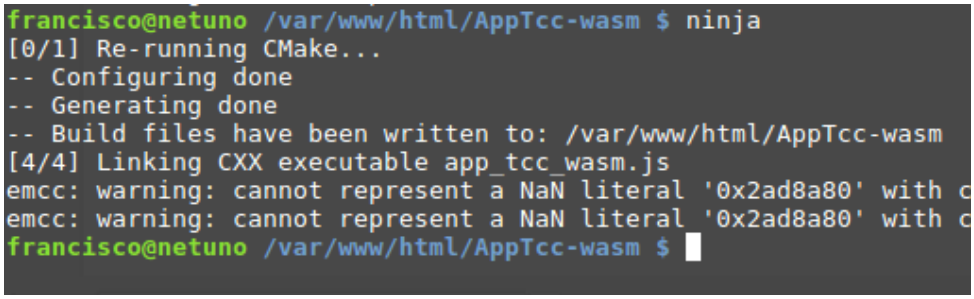
- O CMake fará algumas checagens do ambiente de desenvolvimento. É preciso definir a diretiva “OpenCV_DIR” com o caminho da pasta onde foi gerado o projeto OpenCV (ver APÊNDICE A). Neste exemplo o foi utilizado o diretório “/home/francisco/Documentos/TCC/OpenCV-3.0.0/build”



- Ao final da configuração basta gerar os arquivos que definem a construção do projeto clicando no botão “Generate”



3) Compilar projeto: Após a gerar os arquivos de configuração é o momento de fazer a compilação, executando o comando ”ninja” de dentro da pasta raiz do projeto.



Serão gerados dois novos arquivos como resultado da compilação, o “app_tcc_wasm.js” e o “app_tcc_wasm.wasm”. O “.js” é uma espécie de cola, que contém todo o código responsável por instanciar o módulo WebAssembly a partir do “.wasm”, configurar a memória e tornar disponível as funções escritas em C++.

4) Integração HTML: O exemplo abaixo mostra a integração do .wasm em uma página HTML e um trecho de código JavaScript chamando a função WebAssembly.

```
<!doctype html>
<html>
<body>
<script>

var Module = {
    wasmBinaryFile: 'app_tcc_wasm.wasm',  // define o arquivo .wasm que deve ser carregado

    _main: function() { // função chamada quando módulo WebAssembly esta pronto
        console.log("WASM Pronto");
    },
    _exit: function(x) {
        console.log("fim do programa");
    }
};
</script>

<script type="text/javascript">

    ...

    let homography_matrix;
    let homography_matrix_size = 9; // matrix 3x3
    try {

        // chama a função WebAssembly criada para calcular a homografia
        homography_matrix = Module._vo_homography(img_data.width,      // largura da imagem
                                    img_data.height,    // altura da imagem
                                    fp.frame_bytes.byteOffset, // ponteiro referenciando a imagem
                                    frameIndex, // número do frame
                                    homography_matrix_size); // tamanho do vetor de resultados

    } catch (e) {
        throw e
    }
    // popula o array com o resultado da homografia obtido a partir da função WebAssembly
    var homography = [];
    for (let v = 0; v < homography_matrix_size; v++) {
        homography.push(Module.HEAPF64[homography_matrix / Float64Array.BYTES_PER_ELEMENT + v])
    }

    ...

</script>
<script type="text/javascript" src='app_tcc_wasm.js'></script>
</body>
</html>
```